

jsonToBatProject

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

Mon Feb 26 2024 15:43:45

1 README	1
1.1 README	1
1.1.1 Current workflows:	1
1.1.2 Regarding coding style (?):	1
1.1.3 Git (?):	1
2 Todo List	3
3 Namespace Index	5
3.1 Namespace List	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Data Structure Index	9
5.1 Data Structures	9
6 File Index	13
6.1 File List	13
7 Namespace Documentation	15
7.1 el Namespace Reference	15
7.1.1 Detailed Description	16
7.1.2 Typedef Documentation	17
7.1.2.1 FormatSpecifierValueResolver	17
7.1.2.2 LogBuilderPtr	17
7.1.2.3 PreRollOutCallback	17
7.1.3 Enumeration Type Documentation	17
7.1.3.1 ConfigurationType	17
7.1.3.2 Level	18
7.1.3.3 LoggingFlag	18
7.1.4 Variable Documentation	19
7.1.4.1 configStringToTypeMap	19
7.1.4.2 elCrashHandler	19
7.1.4.3 stringToLevelMap	19
7.2 el::base Namespace Reference	20
7.2.1 Detailed Description	21
7.2.2 Typedef Documentation	21
7.2.2.1 FileStreamPtr	21
7.2.2.2 LogStreamsReferenceMap	21
7.2.2.3 LogStreamsReferenceMapPtr	22
7.2.2.4 MillisecondsWidth	22
7.2.3 Enumeration Type Documentation	22
7.2.3.1 DispatchAction	22

7.2.3.2 FormatFlags	22
7.2.3.3 TimestampUnit	23
7.2.4 Function Documentation	23
7.2.4.1 defaultPreRollOutCallback()	23
7.2.5 Variable Documentation	23
7.2.5.1 elStorage	23
7.3 el::base::consts Namespace Reference	23
7.3.1 Detailed Description	25
7.3.2 Variable Documentation	25
7.3.2.1 brief	25
7.3.2.2 detail	25
7.3.2.3 kAm	25
7.3.2.4 kAppNameFormatSpecifier	26
7.3.2.5 kConfigurationComment	26
7.3.2.6 kConfigurationLevel	26
7.3.2.7 kConfigurationLoggerId	26
7.3.2.8 [struct]	26
7.3.2.9 kCrashSignalsCount	27
7.3.2.10 kCurrentHostFormatSpecifier	27
7.3.2.11 kCurrentUserFormatSpecifier	27
7.3.2.12 kDateTimeFormatSpecifier	27
7.3.2.13 kDateTimeFormatSpecifierForFilename	27
7.3.2.14 kDays	27
7.3.2.15 kDaysAbbrev	27
7.3.2.16 kDebugLevelLogValue	28
7.3.2.17 kDebugLevelShortLogValue	28
7.3.2.18 kDefaultDateTimeFormat	28
7.3.2.19 kDefaultDateTimeFormatInFilename	28
7.3.2.20 kDefaultLogFile	28
7.3.2.21 kDefaultLogFileParam	28
7.3.2.22 kDefaultLoggerId	28
7.3.2.23 kDefaultSubsecondPrecision	28
7.3.2.24 kErrorLevelLogValue	29
7.3.2.25 kErrorLevelShortLogValue	29
7.3.2.26 kFatalLevelLogValue	29
7.3.2.27 kFatalLevelShortLogValue	29
7.3.2.28 kFilePathSeparator	29
7.3.2.29 kFormatSpecifierChar	29
7.3.2.30 kFormatSpecifierCharValue	29
7.3.2.31 kInfoLevelLogValue	29
7.3.2.32 kInfoLevelShortLogValue	30
7.3.2.33 kLogFileBaseFormatSpecifier	30

7.3.2.34 kLogFileFormatSpecifier	30
7.3.2.35 kLogFunctionFormatSpecifier	30
7.3.2.36 kLoggerIdFormatSpecifier	30
7.3.2.37 kLogLineFormatSpecifier	30
7.3.2.38 kLogLocationFormatSpecifier	30
7.3.2.39 kMaxLogPerContainer	31
7.3.2.40 kMaxLogPerCounter	31
7.3.2.41 kMaxVerboseLevel	31
7.3.2.42 kMessageFormatSpecifier	31
7.3.2.43 kMonths	31
7.3.2.44 kMonthsAbbrev	31
7.3.2.45 kNullPointer	31
7.3.2.46 kPerformanceTrackerDefaultLevel	32
7.3.2.47 kPm	32
7.3.2.48 kSeverityLevelFormatSpecifier	32
7.3.2.49 kSeverityLevelShortFormatSpecifier	32
7.3.2.50 kSourceFilenameMaxLength	32
7.3.2.51 kSourceLineMaxLength	32
7.3.2.52 kThreadIdFormatSpecifier	32
7.3.2.53 [struct]	33
7.3.2.54 kTimeFormatsCount	33
7.3.2.55 kTraceLevelLogValue	33
7.3.2.56 kTraceLevelShortLogValue	33
7.3.2.57 kUnknownHost	33
7.3.2.58 kUnknownUser	33
7.3.2.59 kValidLoggerIdSymbols	33
7.3.2.60 kVerboseLevelFormatSpecifier	34
7.3.2.61 kVerboseLevelLogValue	34
7.3.2.62 kVerboseLevelShortLogValue	34
7.3.2.63 kWarningLevelLogValue	34
7.3.2.64 kWarningLevelShortLogValue	34
7.3.2.65 kYearBase	34
7.3.2.66 name	34
7.3.2.67 numb	35
7.3.2.68 unit	35
7.3.2.69 value	35
7.4 el::base::debug Namespace Reference	35
7.4.1 Detailed Description	35
7.5 el::base::threading Namespace Reference	35
7.5.1 Typedef Documentation	36
7.5.1.1 Mutex	36
7.5.1.2 ScopedLock	36

7.5.2 Function Documentation	36
7.5.2.1 getCurrentThreadId()	36
7.6 el::base::threading::internal Namespace Reference	36
7.7 el::base::type Namespace Reference	37
7.7.1 Detailed Description	37
7.7.2 Typedef Documentation	37
7.7.2.1 char_t	37
7.7.2.2 EnumType	37
7.7.2.3 fstream_t	37
7.7.2.4 LineNumber	38
7.7.2.5 LogDispatchCallbackPtr	38
7.7.2.6 LoggerRegistrationCallbackPtr	38
7.7.2.7 ostream_t	38
7.7.2.8 PerformanceTrackerPtr	38
7.7.2.9 PerformanceTrackingCallbackPtr	38
7.7.2.10 StoragePointer	38
7.7.2.11 string_t	38
7.7.2.12 stringstream_t	39
7.7.2.13 VerboseLevel	39
7.8 el::base::utils Namespace Reference	39
7.8.1 Detailed Description	40
7.8.2 Function Documentation	40
7.8.2.1 abort()	40
7.8.2.2 addFlag()	40
7.8.2.3 hasFlag()	40
7.8.2.4 operator<<()	41
7.8.2.5 removeFlag()	41
7.8.2.6 safeDelete()	41
7.9 el::base::utils::bitwise Namespace Reference	41
7.9.1 Detailed Description	41
7.9.2 Function Documentation	42
7.9.2.1 And()	42
7.9.2.2 Not()	42
7.9.2.3 Or()	42
7.10 std Namespace Reference	42
7.11 WIP Namespace Reference	42
7.11.1 Detailed Description	43
7.11.2 Function Documentation	43
7.11.2.1 exampleEasyLogging()	43
8 Data Structure Documentation	45
8.1 el::base::utils::AbstractRegistry< T_Ptr, Container > Class Template Reference	45

8.1.1 Detailed Description	46
8.1.2 Member Typedef Documentation	46
8.1.2.1 const_iterator	46
8.1.2.2 iterator	46
8.1.3 Constructor & Destructor Documentation	47
8.1.3.1 AbstractRegistry() [1/2]	47
8.1.3.2 AbstractRegistry() [2/2]	47
8.1.3.3 ~AbstractRegistry()	47
8.1.4 Member Function Documentation	47
8.1.4.1 begin()	47
8.1.4.2 cbegin()	48
8.1.4.3 cend()	48
8.1.4.4 deepCopy()	48
8.1.4.5 empty()	48
8.1.4.6 end()	49
8.1.4.7 list() [1/2]	49
8.1.4.8 list() [2/2]	49
8.1.4.9 operator!=(())	49
8.1.4.10 operator=()	49
8.1.4.11 operator==(())	50
8.1.4.12 reinitDeepCopy()	50
8.1.4.13 size()	50
8.1.4.14 unregisterAll()	50
8.1.5 Field Documentation	50
8.1.5.1 m_list	50
8.2 el::Callback< T > Class Template Reference	51
8.2.1 Detailed Description	51
8.2.2 Constructor & Destructor Documentation	52
8.2.2.1 Callback()	52
8.2.3 Member Function Documentation	52
8.2.3.1 enabled()	52
8.2.3.2 handle()	52
8.2.3.3 setEnabled()	52
8.2.4 Field Documentation	52
8.2.4.1 m_enabled	52
8.3 el::base::utils::CommandLineArgs Class Reference	53
8.3.1 Detailed Description	53
8.3.2 Constructor & Destructor Documentation	53
8.3.2.1 CommandLineArgs() [1/3]	53
8.3.2.2 CommandLineArgs() [2/3]	54
8.3.2.3 CommandLineArgs() [3/3]	54
8.3.2.4 ~CommandLineArgs()	54

8.3.3 Member Function Documentation	54
8.3.3.1 empty()	54
8.3.3.2 getParamValue()	54
8.3.3.3 hasParam()	55
8.3.3.4 hasParamWithValue()	55
8.3.3.5 setArgs() [1/2]	55
8.3.3.6 setArgs() [2/2]	55
8.3.3.7 size()	55
8.3.4 Friends And Related Symbol Documentation	56
8.3.4.1 operator<<	56
8.3.5 Field Documentation	56
8.3.5.1 m_argc	56
8.3.5.2 m_argv	56
8.3.5.3 m_params	56
8.3.5.4 m_paramsWithValue	56
8.4 el::Configuration Class Reference	57
8.4.1 Detailed Description	58
8.4.2 Constructor & Destructor Documentation	58
8.4.2.1 Configuration() [1/2]	58
8.4.2.2 ~Configuration()	58
8.4.2.3 Configuration() [2/2]	58
8.4.3 Member Function Documentation	58
8.4.3.1 configurationType()	58
8.4.3.2 level()	59
8.4.3.3 log()	59
8.4.3.4 operator=()	59
8.4.3.5 setValue()	59
8.4.3.6 value()	59
8.4.4 Field Documentation	60
8.4.4.1 m_configurationType	60
8.4.4.2 m_level	60
8.4.4.3 m_value	60
8.5 el::Configurations Class Reference	60
8.5.1 Detailed Description	63
8.5.2 Constructor & Destructor Documentation	64
8.5.2.1 Configurations() [1/2]	64
8.5.2.2 Configurations() [2/2]	64
8.5.2.3 ~Configurations()	64
8.5.3 Member Function Documentation	64
8.5.3.1 clear()	64
8.5.3.2 configurationFile()	65
8.5.3.3 get()	65

8.5.3.4 hasConfiguration() [1/2]	65
8.5.3.5 hasConfiguration() [2/2]	65
8.5.3.6 parseFromFile()	66
8.5.3.7 parseFromText()	66
8.5.3.8 set() [1/2]	67
8.5.3.9 set() [2/2]	67
8.5.3.10 setFromBase()	68
8.5.3.11 setGlobally() [1/2]	68
8.5.3.12 setGlobally() [2/2]	68
8.5.3.13 setRemainingToDefault()	69
8.5.3.14 setToDefault()	69
8.5.3.15 unsafeSet()	69
8.5.3.16 unsafeSetGlobally()	70
8.5.3.17 unsafeSetIfNotExist()	70
8.5.4 Friends And Related Symbol Documentation	70
8.5.4.1 el::Loggers	70
8.5.5 Field Documentation	70
8.5.5.1 m_configurationFile	70
8.5.5.2 m_isFromFile	71
8.6 el::ConfigurationStringToTypeltem Struct Reference	71
8.6.1 Detailed Description	71
8.6.2 Field Documentation	71
8.6.2.1 configString	71
8.6.2.2 configType	71
8.7 el::ConfigurationTypeHelper Class Reference	72
8.7.1 Detailed Description	72
8.7.2 Member Function Documentation	72
8.7.2.1 castFromInt()	72
8.7.2.2 castToInt()	73
8.7.2.3 convertFromString()	73
8.7.2.4 convertToString()	73
8.7.2.5 forEachConfigType()	73
8.7.3 Field Documentation	74
8.7.3.1 kMaxValid	74
8.7.3.2 kMinValid	74
8.8 el::base::debug::CrashHandler Class Reference	74
8.8.1 Detailed Description	74
8.8.2 Constructor & Destructor Documentation	75
8.8.2.1 CrashHandler()	75
8.9 el::CustomFormatSpecifier Class Reference	75
8.9.1 Detailed Description	75
8.9.2 Constructor & Destructor Documentation	76

8.9.2.1 CustomFormatSpecifier()	76
8.9.3 Member Function Documentation	76
8.9.3.1 formatSpecifier()	76
8.9.3.2 operator==()	76
8.9.3.3 resolver()	76
8.9.4 Field Documentation	76
8.9.4.1 m_formatSpecifier	76
8.9.4.2 m_resolver	76
8.10 el::base::utils::DateTime Class Reference	77
8.10.1 Detailed Description	77
8.10.2 Member Function Documentation	77
8.10.2.1 buildTimeInfo()	77
8.10.2.2 formatTime()	78
8.10.2.3 getDateTime()	78
8.10.2.4 getTimeDifference()	78
8.10.2.5 gettimeofday()	78
8.10.2.6 parseFormat()	79
8.10.2.7 timevalToString()	79
8.11 el::base::DefaultLogBuilder Class Reference	79
8.11.1 Detailed Description	80
8.11.2 Member Function Documentation	80
8.11.2.1 build()	80
8.12 el::base::DefaultLogDispatchCallback Class Reference	81
8.12.1 Detailed Description	82
8.12.2 Member Function Documentation	82
8.12.2.1 dispatch()	82
8.12.2.2 handle()	82
8.12.3 Field Documentation	82
8.12.3.1 m_data	82
8.13 el::base::utils::File Class Reference	83
8.13.1 Detailed Description	83
8.13.2 Member Function Documentation	83
8.13.2.1 buildBaseFilename()	83
8.13.2.2 buildStrippedFilename()	84
8.13.2.3 createPath()	84
8.13.2.4 extractPathFromFilename()	84
8.13.2.5 getsizeofFile()	84
8.13.2.6 newFileStream()	85
8.13.2.7 pathExists()	85
8.14 std::hash< el::Level > Struct Reference	85
8.14.1 Detailed Description	85
8.14.2 Member Function Documentation	86

8.14.2.1 operator()	86
8.15 el::Helpers Class Reference	86
8.15.1 Detailed Description	87
8.15.2 Member Function Documentation	87
8.15.2.1 commandLineArgs()	87
8.15.2.2 convertTemplateToStdString()	87
8.15.2.3 getThreadName()	88
8.15.2.4 hasCustomFormatSpecifier()	88
8.15.2.5 installCustomFormatSpecifier()	88
8.15.2.6 installLogDispatchCallback()	88
8.15.2.7 installPreRollOutCallback()	88
8.15.2.8 logDispatchCallback()	89
8.15.2.9 reserveCustomFormatSpecifiers()	89
8.15.2.10 setArgs() [1/2]	89
8.15.2.11 setArgs() [2/2]	89
8.15.2.12 setStorage()	90
8.15.2.13 setThreadName()	90
8.15.2.14 storage()	90
8.15.2.15 uninstallCustomFormatSpecifier()	90
8.15.2.16 uninstallLogDispatchCallback()	91
8.15.2.17 uninstallPreRollOutCallback()	91
8.15.2.18 validateFileRolling()	91
8.16 el::base::HitCounter Class Reference	91
8.16.1 Detailed Description	92
8.16.2 Constructor & Destructor Documentation	92
8.16.2.1 HitCounter() [1/3]	92
8.16.2.2 HitCounter() [2/3]	92
8.16.2.3 HitCounter() [3/3]	93
8.16.2.4 ~HitCounter()	93
8.16.3 Member Function Documentation	93
8.16.3.1 filename()	93
8.16.3.2 hitCounts()	93
8.16.3.3 increment()	93
8.16.3.4 lineNumber()	93
8.16.3.5 operator=()	94
8.16.3.6 resetLocation()	94
8.16.3.7 validateHitCounts()	94
8.16.4 Field Documentation	94
8.16.4.1 m_filename	94
8.16.4.2 m_hitCounts	94
8.16.4.3 m_lineNumber	94
8.17 el::LevelHelper Class Reference	95

8.17.1 Detailed Description	95
8.17.2 Member Function Documentation	95
8.17.2.1 castFromInt()	95
8.17.2.2 castToInt()	96
8.17.2.3 convertFromString()	96
8.17.2.4 convertToString()	96
8.17.2.5 forEachLevel()	96
8.17.3 Field Documentation	97
8.17.3.1 kMaxValid	97
8.17.3.2 kMinValid	97
8.18 el::LogBuilder Class Reference	97
8.18.1 Detailed Description	98
8.18.2 Constructor & Destructor Documentation	98
8.18.2.1 LogBuilder()	98
8.18.2.2 ~LogBuilder()	98
8.18.3 Member Function Documentation	99
8.18.3.1 build()	99
8.18.3.2 convertToColoredOutput()	99
8.18.4 Friends And Related Symbol Documentation	99
8.18.4.1 el::base::DefaultLogDispatchCallback	99
8.18.5 Field Documentation	99
8.18.5.1 m_termSupportsColor	99
8.19 el::LogDispatchCallback Class Reference	100
8.19.1 Detailed Description	101
8.19.2 Member Function Documentation	101
8.19.2.1 fileHandle()	101
8.19.2.2 handle()	101
8.19.3 Friends And Related Symbol Documentation	101
8.19.3.1 base::LogDispatcher	101
8.19.4 Field Documentation	101
8.19.4.1 m_fileLocks	101
8.19.4.2 m_fileLocksMapLock	102
8.20 el::LogDispatchData Class Reference	102
8.20.1 Detailed Description	102
8.20.2 Constructor & Destructor Documentation	102
8.20.2.1 LogDispatchData()	102
8.20.3 Member Function Documentation	103
8.20.3.1 dispatchAction()	103
8.20.3.2 logMessage()	103
8.20.3.3 setDispatchAction()	103
8.20.3.4 setLogMessage()	103
8.20.4 Friends And Related Symbol Documentation	103

8.20.4.1 base::LogDispatcher	103
8.20.5 Field Documentation	103
8.20.5.1 m_dispatchAction	103
8.20.5.2 m_logMessage	104
8.21 el::base::LogDispatcher Class Reference	104
8.21.1 Detailed Description	104
8.21.2 Constructor & Destructor Documentation	105
8.21.2.1 LogDispatcher()	105
8.21.3 Member Function Documentation	105
8.21.3.1 dispatch()	105
8.21.4 Field Documentation	105
8.21.4.1 m_dispatchAction	105
8.21.4.2 m_logMessage	105
8.21.4.3 m_proceed	105
8.22 el::base::LogFormat Class Reference	106
8.22.1 Detailed Description	107
8.22.2 Constructor & Destructor Documentation	107
8.22.2.1 LogFormat() [1/4]	107
8.22.2.2 LogFormat() [2/4]	107
8.22.2.3 LogFormat() [3/4]	107
8.22.2.4 LogFormat() [4/4]	107
8.22.2.5 ~LogFormat()	108
8.22.3 Member Function Documentation	108
8.22.3.1 addFlag()	108
8.22.3.2 dateTimeFormat()	108
8.22.3.3 flags()	108
8.22.3.4 format()	108
8.22.3.5 hasFlag()	108
8.22.3.6 level()	109
8.22.3.7 log()	109
8.22.3.8 operator=()	109
8.22.3.9 operator==()	109
8.22.3.10 parseFromFormat()	109
8.22.3.11 updateDateFormat()	110
8.22.3.12 updateFormatSpec()	110
8.22.3.13 userFormat()	110
8.22.4 Friends And Related Symbol Documentation	111
8.22.4.1 el::Logger	111
8.22.5 Field Documentation	111
8.22.5.1 m_currentHost	111
8.22.5.2 m_currentUser	111
8.22.5.3 m_dateTimeFormat	111

8.22.5.4 m_flags	111
8.22.5.5 m_format	111
8.22.5.6 m_level	111
8.22.5.7 m_userFormat	112
8.23 el::Loggable Class Reference	112
8.23.1 Detailed Description	112
8.23.2 Constructor & Destructor Documentation	112
8.23.2.1 ~Loggable()	112
8.23.3 Member Function Documentation	113
8.23.3.1 log()	113
8.23.4 Friends And Related Symbol Documentation	113
8.23.4.1 operator<<	113
8.24 el::Logger Class Reference	113
8.24.1 Detailed Description	115
8.24.2 Constructor & Destructor Documentation	115
8.24.2.1 Logger() [1/4]	115
8.24.2.2 Logger() [2/4]	116
8.24.2.3 Logger() [3/4]	116
8.24.2.4 ~Logger()	116
8.24.2.5 Logger() [4/4]	116
8.24.3 Member Function Documentation	116
8.24.3.1 configurations()	116
8.24.3.2 configure()	116
8.24.3.3 enabled()	117
8.24.3.4 flush() [1/2]	117
8.24.3.5 flush() [2/2]	117
8.24.3.6 id()	117
8.24.3.7 initUnflushedCount()	117
8.24.3.8 isFlushNeeded()	118
8.24.3.9 isValidId()	118
8.24.3.10 log()	118
8.24.3.11 logBuilder()	118
8.24.3.12 operator=()	118
8.24.3.13 parentApplicationName()	118
8.24.3.14 reconfigure()	119
8.24.3.15 resolveLoggerFormatSpec()	119
8.24.3.16 setLogBuilder()	119
8.24.3.17 setParentApplicationName()	119
8.24.3.18 stream()	119
8.24.3.19 typedConfigurations()	119
8.24.4 Friends And Related Symbol Documentation	120
8.24.4.1 el::base::DefaultLogDispatchCallback	120

8.24.4.2 el::base::LogDispatcher	120
8.24.4.3 el::base::MessageBuilder	120
8.24.4.4 el::base::PerformanceTracker	120
8.24.4.5 el::base::PErrorWriter	120
8.24.4.6 el::base::RegisteredLoggers	120
8.24.4.7 el::base::Storage	120
8.24.4.8 el::base::Writer	121
8.24.4.9 el::Helpers	121
8.24.4.10 el::Loggers	121
8.24.4.11 el::LogMessage	121
8.24.5 Field Documentation	121
8.24.5.1 m_configurations	121
8.24.5.2 m_id	121
8.24.5.3 m_isConfigured	121
8.24.5.4 m_logBuilder	122
8.24.5.5 m_logStreamsReference	122
8.24.5.6 m_parentApplicationName	122
8.24.5.7 m_stream	122
8.24.5.8 m_typedConfigurations	122
8.24.5.9 m_unflushedCount	122
8.25 el::LoggerRegistrationCallback Class Reference	123
8.25.1 Detailed Description	123
8.25.2 Friends And Related Symbol Documentation	124
8.25.2.1 base::RegisteredLoggers	124
8.26 el::Loggers Class Reference	124
8.26.1 Detailed Description	126
8.26.2 Member Function Documentation	126
8.26.2.1 addFlag()	126
8.26.2.2 clearVModules()	126
8.26.2.3 configureFromArg()	126
8.26.2.4 configureFromGlobal()	127
8.26.2.5 defaultConfigurations()	127
8.26.2.6 defaultTypedConfigurations()	127
8.26.2.7 flushAll()	127
8.26.2.8 getLogger()	128
8.26.2.9 hasFlag()	128
8.26.2.10 hasLogger()	128
8.26.2.11 installLoggerRegistrationCallback()	128
8.26.2.12 loggerRegistrationCallback()	128
8.26.2.13 logStreamsReference()	129
8.26.2.14 populateAllLoggerIds()	129
8.26.2.15 reconfigureAllLoggers() [1/3]	129

8.26.2.16 reconfigureAllLoggers() [2/3]	129
8.26.2.17 reconfigureAllLoggers() [3/3]	130
8.26.2.18 reconfigureLogger() [1/3]	130
8.26.2.19 reconfigureLogger() [2/3]	130
8.26.2.20 reconfigureLogger() [3/3]	130
8.26.2.21 removeFlag()	131
8.26.2.22 setDefaultConfigurations()	131
8.26.2.23 setDefaultLogBuilder()	131
8.26.2.24 setLoggingLevel()	131
8.26.2.25 setVerboseLevel()	131
8.26.2.26 setVModules()	132
8.26.2.27 uninstallLoggerRegistrationCallback()	132
8.26.2.28 unregisterLogger()	132
8.26.2.29 verboseLevel()	132
8.27 el::LogMessage Class Reference	133
8.27.1 Detailed Description	133
8.27.2 Constructor & Destructor Documentation	133
8.27.2.1 LogMessage()	133
8.27.3 Member Function Documentation	134
8.27.3.1 file()	134
8.27.3.2 func()	134
8.27.3.3 level()	134
8.27.3.4 line()	134
8.27.3.5 logger()	134
8.27.3.6 message()	134
8.27.3.7 verboseLevel()	135
8.27.4 Field Documentation	135
8.27.4.1 m_file	135
8.27.4.2 m_func	135
8.27.4.3 m_level	135
8.27.4.4 m_line	135
8.27.4.5 m_logger	135
8.27.4.6 m_message	135
8.27.4.7 m_verboseLevel	136
8.28 el::base::MessageBuilder Class Reference	136
8.28.1 Detailed Description	136
8.28.2 Constructor & Destructor Documentation	136
8.28.2.1 MessageBuilder()	136
8.28.3 Member Function Documentation	137
8.28.3.1 initialize()	137
8.28.3.2 operator<<() [1/4]	137
8.28.3.3 operator<<() [2/4]	137

8.28.3.4 operator<<() [3/4]	137
8.28.3.5 operator<<() [4/4]	137
8.28.3.6 writeIterator()	137
8.28.4 Field Documentation	138
8.28.4.1 m_containerLogSeparator	138
8.28.4.2 m_logger	138
8.29 el::base::NoCopy Class Reference	138
8.29.1 Detailed Description	139
8.29.2 Constructor & Destructor Documentation	139
8.29.2.1 NoCopy() [1/2]	139
8.29.2.2 NoCopy() [2/2]	139
8.29.3 Member Function Documentation	139
8.29.3.1 operator=()	139
8.30 el::base::threading::internal::NoMutex Class Reference	139
8.30.1 Detailed Description	140
8.30.2 Constructor & Destructor Documentation	140
8.30.2.1 NoMutex()	140
8.30.3 Member Function Documentation	140
8.30.3.1 lock()	140
8.30.3.2 try_lock()	140
8.30.3.3 unlock()	141
8.31 el::base::threading::internal::NoScopedLock< Mutex > Class Template Reference	141
8.31.1 Detailed Description	141
8.31.2 Constructor & Destructor Documentation	142
8.31.2.1 NoScopedLock() [1/2]	142
8.31.2.2 ~NoScopedLock()	142
8.31.2.3 NoScopedLock() [2/2]	142
8.32 el::base::NullWriter Class Reference	142
8.32.1 Detailed Description	143
8.32.2 Constructor & Destructor Documentation	143
8.32.2.1 NullWriter()	143
8.32.3 Member Function Documentation	143
8.32.3.1 operator bool()	143
8.32.3.2 operator<<() [1/2]	143
8.32.3.3 operator<<() [2/2]	143
8.33 el::base::utils::OS Class Reference	144
8.33.1 Detailed Description	144
8.33.2 Member Function Documentation	144
8.33.2.1 currentHost()	144
8.33.2.2 currentUser()	145
8.33.2.3 getBashOutput()	145
8.33.2.4 getEnvironmentVariable()	145

8.33.2.5 termSupportsColor()	146
8.34 el::Configurations::Parser Class Reference	146
8.34.1 Detailed Description	147
8.34.2 Member Function Documentation	147
8.34.2.1 ignoreComments()	147
8.34.2.2 isComment()	147
8.34.2.3 isConfig()	147
8.34.2.4 isLevel()	147
8.34.2.5 parseFromFile()	147
8.34.2.6 parseFromText()	148
8.34.2.7 parseLine()	149
8.34.3 Friends And Related Symbol Documentation	149
8.34.3.1 el::Loggers	149
8.35 el::PerformanceTrackingCallback Class Reference	149
8.35.1 Detailed Description	150
8.35.2 Friends And Related Symbol Documentation	150
8.35.2.1 base::PerformanceTracker	150
8.36 el::base::PErrorWriter Class Reference	150
8.36.1 Detailed Description	151
8.36.2 Constructor & Destructor Documentation	152
8.36.2.1 PErrorWriter()	152
8.36.2.2 ~PErrorWriter()	152
8.37 el::base::HitCounter::Predicate Class Reference	152
8.37.1 Detailed Description	152
8.37.2 Constructor & Destructor Documentation	153
8.37.2.1 Predicate()	153
8.37.3 Member Function Documentation	153
8.37.3.1 operator()()	153
8.37.4 Field Documentation	153
8.37.4.1 m_filename	153
8.37.4.2 m_lineNumber	153
8.38 el::Configuration::Predicate Class Reference	153
8.38.1 Detailed Description	154
8.38.2 Constructor & Destructor Documentation	154
8.38.2.1 Predicate()	154
8.38.3 Member Function Documentation	154
8.38.3.1 operator()()	154
8.38.4 Field Documentation	154
8.38.4.1 m_configurationType	154
8.38.4.2 m_level	155
8.39 el::base::RegisteredHitCounters Class Reference	155
8.39.1 Detailed Description	157

8.39.2 Member Function Documentation	157
8.39.2.1 getCounter()	157
8.39.2.2 validateAfterN()	157
8.39.2.3 validateEveryN()	158
8.39.2.4 validateNTimes()	158
8.40 el::base::RegisteredLoggers Class Reference	158
8.40.1 Detailed Description	161
8.40.2 Constructor & Destructor Documentation	161
8.40.2.1 RegisteredLoggers()	161
8.40.2.2 ~RegisteredLoggers()	161
8.40.3 Member Function Documentation	161
8.40.3.1 defaultConfigurations()	161
8.40.3.2 flushAll()	162
8.40.3.3 get()	162
8.40.3.4 has()	162
8.40.3.5 installLoggerRegistrationCallback()	162
8.40.3.6 loggerRegistrationCallback()	162
8.40.3.7 logStreamsReference()	162
8.40.3.8 remove()	163
8.40.3.9 setDefaultConfigurations()	163
8.40.3.10 setDefaultLogBuilder()	163
8.40.3.11 uninstallLoggerRegistrationCallback()	163
8.40.3.12 unregister()	163
8.40.3.13 unsafeFlushAll()	163
8.40.4 Friends And Related Symbol Documentation	164
8.40.4.1 el::base::Storage	164
8.40.5 Field Documentation	164
8.40.5.1 m_defaultConfigurations	164
8.40.5.2 m_defaultLogBuilder	164
8.40.5.3 m_loggerRegistrationCallbacks	164
8.40.5.4 m_logStreamsReference	164
8.41 el::base::utils::Registry< T_Ptr, T_Key > Class Template Reference	165
8.41.1 Detailed Description	166
8.41.2 Member Typedef Documentation	167
8.41.2.1 const_iterator	167
8.41.2.2 iterator	167
8.41.3 Constructor & Destructor Documentation	167
8.41.3.1 Registry() [1/2]	167
8.41.3.2 Registry() [2/2]	167
8.41.3.3 ~Registry()	167
8.41.4 Member Function Documentation	168
8.41.4.1 deepCopy()	168

8.41.4.2	get()	168
8.41.4.3	operator=()	168
8.41.4.4	registerNew()	168
8.41.4.5	unregister()	169
8.41.4.6	unregisterAll()	169
8.42	el::base::utils::RegistryWithPred< T_Ptr, Pred > Class Template Reference	169
8.42.1	Detailed Description	171
8.42.2	Member Typedef Documentation	171
8.42.2.1	const_iterator	171
8.42.2.2	iterator	171
8.42.3	Constructor & Destructor Documentation	172
8.42.3.1	RegistryWithPred() [1/2]	172
8.42.3.2	~RegistryWithPred()	172
8.42.3.3	RegistryWithPred() [2/2]	172
8.42.4	Member Function Documentation	172
8.42.4.1	deepCopy()	172
8.42.4.2	get()	173
8.42.4.3	operator=()	173
8.42.4.4	registerNew()	173
8.42.4.5	unregister()	173
8.42.4.6	unregisterAll()	173
8.42.5	Friends And Related Symbol Documentation	174
8.42.5.1	operator<<	174
8.43	el::Loggers::ScopedAddFlag Class Reference	174
8.43.1	Detailed Description	174
8.43.2	Constructor & Destructor Documentation	174
8.43.2.1	ScopedAddFlag()	174
8.43.2.2	~ScopedAddFlag()	175
8.43.3	Field Documentation	175
8.43.3.1	m_flag	175
8.44	el::Loggers::ScopedRemoveFlag Class Reference	175
8.44.1	Detailed Description	175
8.44.2	Constructor & Destructor Documentation	175
8.44.2.1	ScopedRemoveFlag()	175
8.44.2.2	~ScopedRemoveFlag()	176
8.44.3	Field Documentation	176
8.44.3.1	m_flag	176
8.45	el::base::StaticClass Class Reference	176
8.45.1	Detailed Description	177
8.45.2	Constructor & Destructor Documentation	177
8.45.2.1	StaticClass() [1/2]	177
8.45.2.2	StaticClass() [2/2]	177

8.45.3 Member Function Documentation	177
8.45.3.1 operator=()	177
8.46 el::base::Storage Class Reference	177
8.46.1 Detailed Description	179
8.46.2 Constructor & Destructor Documentation	179
8.46.2.1 Storage()	179
8.46.2.2 ~Storage()	180
8.46.3 Member Function Documentation	180
8.46.3.1 addFlag()	180
8.46.3.2 commandLineArgs()	180
8.46.3.3 customFormatSpecifiers()	180
8.46.3.4 customFormatSpecifiersLock()	180
8.46.3.5 flags()	180
8.46.3.6 getThreadName()	181
8.46.3.7 hasCustomFormatSpecifier()	181
8.46.3.8 hasFlag()	181
8.46.3.9 hitCounters()	181
8.46.3.10 installCustomFormatSpecifier()	181
8.46.3.11 installLogDispatchCallback()	181
8.46.3.12 logDispatchCallback()	182
8.46.3.13 preRollOutCallback()	182
8.46.3.14 registeredLoggers()	182
8.46.3.15 removeFlag()	182
8.46.3.16 setApplicationArguments() [1/2]	182
8.46.3.17 setApplicationArguments() [2/2]	182
8.46.3.18 setFlags()	183
8.46.3.19 setLoggingLevel()	183
8.46.3.20 setPreRollOutCallback()	183
8.46.3.21 setThreadName()	183
8.46.3.22 uninstallCustomFormatSpecifier()	183
8.46.3.23 uninstallLogDispatchCallback()	183
8.46.3.24 unsetPreRollOutCallback()	184
8.46.3.25 validateAfterNCounter()	184
8.46.3.26 validateEveryNCounter()	184
8.46.3.27 validateNTimesCounter()	184
8.46.3.28 vRegistry()	184
8.46.4 Friends And Related Symbol Documentation	184
8.46.4.1 el::base::DefaultLogDispatchCallback	184
8.46.4.2 el::base::LogDispatcher	185
8.46.4.3 el::base::MessageBuilder	185
8.46.4.4 el::base::PerformanceTracker	185
8.46.4.5 el::base::Writer	185

8.46.4.6 el::Helpers	185
8.46.4.7 el::LogBuilder	185
8.46.5 Field Documentation	185
8.46.5.1 m_commandLineArgs	185
8.46.5.2 m_customFormatSpecifiers	186
8.46.5.3 m_customFormatSpecifiersLock	186
8.46.5.4 m_flags	186
8.46.5.5 m_logDispatchCallbacks	186
8.46.5.6 m_loggingLevel	186
8.46.5.7 m_performanceTrackingCallbacks	186
8.46.5.8 m_preRollOutCallback	186
8.46.5.9 m_registeredHitCounters	187
8.46.5.10 m_registeredLoggers	187
8.46.5.11 m_threadNames	187
8.46.5.12 m_threadNamesLock	187
8.46.5.13 m_vRegistry	187
8.47 el::base::utils::Str Class Reference	187
8.47.1 Detailed Description	188
8.47.2 Member Function Documentation	188
8.47.2.1 addToBuff()	188
8.47.2.2 clearBuff()	189
8.47.2.3 contains()	189
8.47.2.4 convertAndAddToBuff()	189
8.47.2.5 cStringCaseEq()	189
8.47.2.6 cStringEq()	189
8.47.2.7 endsWith()	189
8.47.2.8 isDigit()	190
8.47.2.9 ltrim()	190
8.47.2.10 replaceAll() [1/2]	190
8.47.2.11 replaceAll() [2/2]	191
8.47.2.12 replaceFirstWithEscape()	191
8.47.2.13 rtrim()	191
8.47.2.14 startsWith()	191
8.47.2.15 toUpper()	192
8.47.2.16 trim()	192
8.47.2.17 wcharPtrToCharPtr()	192
8.47.2.18 wildCardMatch()	193
8.48 el::StringToLevelItem Struct Reference	193
8.48.1 Detailed Description	193
8.48.2 Field Documentation	193
8.48.2.1 level	193
8.48.2.2 levelString	193

8.49 <code>el::base::SubsecondPrecision</code> Class Reference	194
8.49.1 Detailed Description	194
8.49.2 Constructor & Destructor Documentation	194
8.49.2.1 <code>SubsecondPrecision()</code> [1/2]	194
8.49.2.2 <code>SubsecondPrecision()</code> [2/2]	194
8.49.3 Member Function Documentation	195
8.49.3.1 <code>init()</code>	195
8.49.3.2 <code>operator==()</code>	195
8.49.4 Field Documentation	195
8.49.4.1 <code>m_offset</code>	195
8.49.4.2 <code>m_width</code>	195
8.50 <code>el::SysLogInitializer</code> Class Reference	195
8.50.1 Detailed Description	196
8.50.2 Constructor & Destructor Documentation	196
8.50.2.1 <code>SysLogInitializer()</code>	196
8.50.2.2 <code>~SysLogInitializer()</code>	196
8.51 <code>el::base::threading::ThreadSafe</code> Class Reference	197
8.51.1 Detailed Description	197
8.51.2 Constructor & Destructor Documentation	198
8.51.2.1 <code>ThreadSafe()</code>	198
8.51.2.2 <code>~ThreadSafe()</code>	198
8.51.3 Member Function Documentation	198
8.51.3.1 <code>acquireLock()</code>	198
8.51.3.2 <code>lock()</code>	198
8.51.3.3 <code>releaseLock()</code>	198
8.51.4 Field Documentation	198
8.51.4.1 <code>m_mutex</code>	198
8.52 <code>el::base::TypedConfigurations</code> Class Reference	199
8.52.1 Detailed Description	200
8.52.2 Constructor & Destructor Documentation	201
8.52.2.1 <code>TypedConfigurations()</code> [1/2]	201
8.52.2.2 <code>TypedConfigurations()</code> [2/2]	202
8.52.2.3 <code>~TypedConfigurations()</code>	202
8.52.3 Member Function Documentation	202
8.52.3.1 <code>build()</code>	202
8.52.3.2 <code>configurations()</code>	202
8.52.3.3 <code>enabled()</code>	203
8.52.3.4 <code>filename()</code>	203
8.52.3.5 <code>fileStream()</code>	203
8.52.3.6 <code>getConfigByRef()</code>	203
8.52.3.7 <code>getConfigByVal()</code>	203
8.52.3.8 <code>getULong()</code>	204

8.52.3.9 insertFile()	204
8.52.3.10 logFlushThreshold()	204
8.52.3.11 logFormat()	204
8.52.3.12 maxLogFileSize()	204
8.52.3.13 millisecondsWidth()	205
8.52.3.14 performanceTracking()	205
8.52.3.15 resolveFilename()	205
8.52.3.16 setValue()	205
8.52.3.17 subsecondPrecision()	205
8.52.3.18 toFile()	206
8.52.3.19 toStandardOutput()	206
8.52.3.20 unsafeGetConfigByRef()	206
8.52.3.21 unsafeGetConfigByVal()	206
8.52.3.22 unsafeValidateFileRolling()	206
8.52.3.23 validateFileRolling()	207
8.52.4 Friends And Related Symbol Documentation	207
8.52.4.1 el::base::DefaultLogDispatchCallback	207
8.52.4.2 el::base::LogDispatcher	207
8.52.4.3 el::base::MessageBuilder	207
8.52.4.4 el::base::Writer	207
8.52.4.5 el::Helpers	207
8.52.5 Field Documentation	207
8.52.5.1 m_configurations	207
8.52.5.2 m_enabledMap	208
8.52.5.3 m_filenameMap	208
8.52.5.4 m_fileStreamMap	208
8.52.5.5 m_logFlushThresholdMap	208
8.52.5.6 m_logFormatMap	208
8.52.5.7 m_logStreamsReference	208
8.52.5.8 m_maxLogFileSizeMap	208
8.52.5.9 m_performanceTrackingMap	209
8.52.5.10 m_subsecondPrecisionMap	209
8.52.5.11 m_toFileMap	209
8.52.5.12 m_toStandardOutputMap	209
8.53 el::base::utils::Utils Class Reference	209
8.53.1 Detailed Description	209
8.53.2 Member Function Documentation	210
8.53.2.1 callback()	210
8.53.2.2 installCallback()	210
8.53.2.3 uninstallCallback()	210
8.54 el::VersionInfo Class Reference	210
8.54.1 Detailed Description	211

8.54.2 Member Function Documentation	211
8.54.2.1 releaseDate()	211
8.54.2.2 version()	211
8.55 el::base::VRegistry Class Reference	211
8.55.1 Detailed Description	212
8.55.2 Constructor & Destructor Documentation	213
8.55.2.1 VRegistry()	213
8.55.3 Member Function Documentation	213
8.55.3.1 allowed()	213
8.55.3.2 clearModules()	213
8.55.3.3 level()	213
8.55.3.4 modules()	213
8.55.3.5 setFromArgs()	214
8.55.3.6 setLevel()	214
8.55.3.7 setModules()	214
8.55.3.8 vModulesEnabled()	214
8.55.4 Field Documentation	214
8.55.4.1 m_level	214
8.55.4.2 m_modules	215
8.55.4.3 m_pFlags	215
8.56 el::base::Writer Class Reference	215
8.56.1 Detailed Description	216
8.56.2 Constructor & Destructor Documentation	216
8.56.2.1 Writer() [1/2]	216
8.56.2.2 Writer() [2/2]	216
8.56.2.3 ~Writer()	217
8.56.3 Member Function Documentation	217
8.56.3.1 construct() [1/2]	217
8.56.3.2 construct() [2/2]	217
8.56.3.3 initializeLogger()	217
8.56.3.4 operator bool()	217
8.56.3.5 operator<<() [1/2]	218
8.56.3.6 operator<<() [2/2]	218
8.56.3.7 processDispatch()	218
8.56.3.8 triggerDispatch()	218
8.56.4 Friends And Related Symbol Documentation	218
8.56.4.1 el::Helpers	218
8.56.5 Field Documentation	218
8.56.5.1 m_dispatchAction	218
8.56.5.2 m_file	219
8.56.5.3 m_func	219
8.56.5.4 m_level	219

8.56.5.5 m_line	219
8.56.5.6 m_logger	219
8.56.5.7 m_loggerIds	219
8.56.5.8 m_messageBuilder	219
8.56.5.9 m_msg	219
8.56.5.10 m_proceed	220
8.56.5.11 m_verboseLevel	220
9 File Documentation	221
9.1 lib/easylogging++.cc File Reference	221
9.1.1 Macro Definition Documentation	223
9.1.1.1 ELPP_DEFAULT_LOGGING_FLAGS	223
9.2 easylogging++.cc	223
9.3 lib/easylogging++.h File Reference	260
9.3.1 Macro Definition Documentation	271
9.3.1.1 CCHECK	271
9.3.1.2 CCHECK_BOUNDS	271
9.3.1.3 CCHECK_EQ	271
9.3.1.4 CCHECK_GE	271
9.3.1.5 CCHECK_GT	272
9.3.1.6 CCHECK_LE	272
9.3.1.7 CCHECK_LT	272
9.3.1.8 CCHECK_NE	272
9.3.1.9 CCHECK_NOTNULL	272
9.3.1.10 CCHECK_STRCASEEQ	273
9.3.1.11 CCHECK_STRCASENE	273
9.3.1.12 CCHECK_STREQ	273
9.3.1.13 CCHECK_STRNE	273
9.3.1.14 CDEBUG	274
9.3.1.15 CDEBUG_AFTER_N	274
9.3.1.16 CDEBUG_EVERY_N	274
9.3.1.17 CDEBUG_IF	274
9.3.1.18 CDEBUG_N_TIMES	274
9.3.1.19 CERROR	275
9.3.1.20 CERROR_AFTER_N	275
9.3.1.21 CERROR_EVERY_N	275
9.3.1.22 CERROR_IF	275
9.3.1.23 CERROR_N_TIMES	275
9.3.1.24 CFATAL	276
9.3.1.25 CFATAL_AFTER_N	276
9.3.1.26 CFATAL_EVERY_N	276
9.3.1.27 CFATAL_IF	276

9.3.1.28 CFATAL_N_TIMES	276
9.3.1.29 CHECK	277
9.3.1.30 CHECK_BOUNDS	277
9.3.1.31 CHECK_EQ	277
9.3.1.32 CHECK_GE	277
9.3.1.33 CHECK_GT	277
9.3.1.34 CHECK_LE	277
9.3.1.35 CHECK_LT	278
9.3.1.36 CHECK_NE	278
9.3.1.37 CHECK_NOTNULL	278
9.3.1.38 CHECK_STRCASEEQ	278
9.3.1.39 CHECK_STRCASENE	278
9.3.1.40 CHECK_STREQ	278
9.3.1.41 CHECK_STRNE	279
9.3.1.42 CINFO	279
9.3.1.43 CINFO_AFTER_N	279
9.3.1.44 CINFO_EVERY_N	279
9.3.1.45 CINFO_IF	279
9.3.1.46 CINFO_N_TIMES	280
9.3.1.47 CLOG	280
9.3.1.48 CLOG_AFTER_N	280
9.3.1.49 CLOG_EVERY_N	280
9.3.1.50 CLOG_IF	280
9.3.1.51 CLOG_N_TIMES	281
9.3.1.52 CPCHECK	281
9.3.1.53 CPLOG	281
9.3.1.54 CPLOG_IF	281
9.3.1.55 CSYSLOG	281
9.3.1.56 CSYSLOG_AFTER_N	282
9.3.1.57 CSYSLOG_EVERY_N	282
9.3.1.58 CSYSLOG_IF	282
9.3.1.59 CSYSLOG_N_TIMES	282
9.3.1.60 CTRACE	282
9.3.1.61 CTRACE_AFTER_N	283
9.3.1.62 CTRACE_EVERY_N	283
9.3.1.63 CTRACE_IF	283
9.3.1.64 CTRACE_N_TIMES	283
9.3.1.65 CVERBOSE	283
9.3.1.66 CVERBOSE_AFTER_N	284
9.3.1.67 CVERBOSE_EVERY_N	284
9.3.1.68 CVERBOSE_IF	284
9.3.1.69 CVERBOSE_N_TIMES	284

9.3.1.70 CVLOG	285
9.3.1.71 CVLOG_AFTER_N	285
9.3.1.72 CVLOG_EVERY_N	285
9.3.1.73 CVLOG_IF	285
9.3.1.74 CVLOG_N_TIMES	285
9.3.1.75 CWARNING	286
9.3.1.76 CWARNING_AFTER_N	286
9.3.1.77 CWARNING_EVERY_N	286
9.3.1.78 CWARNING_IF	286
9.3.1.79 CWARNING_N_TIMES	286
9.3.1.80 DCCHECK	287
9.3.1.81 DCCHECK_BOUNDS	287
9.3.1.82 DCCHECK_EQ	287
9.3.1.83 DCCHECK_GE	287
9.3.1.84 DCCHECK_GT	287
9.3.1.85 DCCHECK_LE	287
9.3.1.86 DCCHECK_LT	288
9.3.1.87 DCCHECK_NE	288
9.3.1.88 DCCHECK_NOTNULL	288
9.3.1.89 DCCHECK_STRCASEEQ	288
9.3.1.90 DCCHECK_STRCASENE	288
9.3.1.91 DCCHECK_STREQ	288
9.3.1.92 DCCHECK_STRNE	289
9.3.1.93 DCHECK	289
9.3.1.94 DCHECK_BOUNDS	289
9.3.1.95 DCHECK_EQ	289
9.3.1.96 DCHECK_GE	289
9.3.1.97 DCHECK_GT	289
9.3.1.98 DCHECK_LE	290
9.3.1.99 DCHECK_LT	290
9.3.1.100 DCHECK_NE	290
9.3.1.101 DCHECK_NOTNULL	290
9.3.1.102 DCHECK_STRCASEEQ	290
9.3.1.103 DCHECK_STRCASENE	290
9.3.1.104 DCHECK_STREQ	291
9.3.1.105 DCHECK_STRNE	291
9.3.1.106 DCLOG	291
9.3.1.107 DCLOG_AFTER_N	291
9.3.1.108 DCLOG_EVERY_N	291
9.3.1.109 DCLOG_IF	291
9.3.1.110 DCLOG_N_TIMES	292
9.3.1.111 DCLOG_VERBOSE	292

9.3.1.112 DCPCHECK	292
9.3.1.113 DCPLOG	292
9.3.1.114 DCPLOG_IF	292
9.3.1.115 DCSYSLOG	292
9.3.1.116 DCSYSLOG_AFTER_N	293
9.3.1.117 DCSYSLOG_EVERY_N	293
9.3.1.118 DCSYSLOG_IF	293
9.3.1.119 DCSYSLOG_N_TIMES	293
9.3.1.120 DCVLOG	293
9.3.1.121 DCVLOG_AFTER_N	293
9.3.1.122 DCVLOG_EVERY_N	294
9.3.1.123 DCVLOG_IF	294
9.3.1.124 DCVLOG_N_TIMES	294
9.3.1.125 DLOG	294
9.3.1.126 DLOG_AFTER_N	294
9.3.1.127 DLOG_EVERY_N	294
9.3.1.128 DLOG_IF	295
9.3.1.129 DLOG_N_TIMES	295
9.3.1.130 DPCHECK	295
9.3.1.131 DPLOG	295
9.3.1.132 DPLOG_IF	295
9.3.1.133 DSYSLOG	295
9.3.1.134 DSYSLOG_AFTER_N	296
9.3.1.135 DSYSLOG_EVERY_N	296
9.3.1.136 DSYSLOG_IF	296
9.3.1.137 DSYSLOG_N_TIMES	296
9.3.1.138 DVLOG	296
9.3.1.139 DVLOG_AFTER_N	296
9.3.1.140 DVLOG_EVERY_N	297
9.3.1.141 DVLOG_IF	297
9.3.1.142 DVLOG_N_TIMES	297
9.3.1.143 el_getVLength	297
9.3.1.144 el_resolveVLength	297
9.3.1.145 ELPP	298
9.3.1.146 ELPP_ASSERT	298
9.3.1.147 ELPP_ASYNC_LOGGING	298
9.3.1.148 ELPP_COMPILER_CLANG	298
9.3.1.149 ELPP_COMPILER_GCC	298
9.3.1.150 ELPP_COMPILER_INTEL	298
9.3.1.151 ELPP_COMPILER_MSVC	299
9.3.1.152 ELPP_COUNTER	299
9.3.1.153 ELPP_COUNTER_POS	299

9.3.1.154 ELPP_COUT	299
9.3.1.155 ELPP_COUT_LINE	299
9.3.1.156 ELPP_CRASH_HANDLER_INIT	299
9.3.1.157 ELPP_CRT_DBG_WARNINGS	299
9.3.1.158 ELPP_CURR_FILE_LOGGER_ID	300
9.3.1.159 ELPP_CYGWIN	300
9.3.1.160 ELPP_DEBUG_LOG	300
9.3.1.161 ELPP_ERROR_LOG	300
9.3.1.162 ELPP_EXPORT	300
9.3.1.163 ELPP_FATAL_LOG	300
9.3.1.164 ELPP_FINAL	300
9.3.1.165 ELPP_FUNC	300
9.3.1.166 ELPP_INFO_LOG	301
9.3.1.167 ELPP_INIT_EASYLOGGINGPP	301
9.3.1.168 ELPP_INITIALIZE_SYSLOG	301
9.3.1.169 ELPP_INTERNAL_DEBUGGING_ENDL	301
9.3.1.170 ELPP_INTERNAL_DEBUGGING_MSG	301
9.3.1.171 ELPP_INTERNAL_DEBUGGING_OUT_ERROR	301
9.3.1.172 ELPP_INTERNAL_DEBUGGING_OUT_INFO	302
9.3.1.173 ELPP_INTERNAL_DEBUGGING_WRITE_PERROR	302
9.3.1.174 ELPP_INTERNAL_ERROR	302
9.3.1.175 ELPP_INTERNAL_INFO	302
9.3.1.176 ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG	302
9.3.1.177 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG	302
9.3.1.178 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG	303
9.3.1.179 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG	303
9.3.1.180 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG	303
9.3.1.181 ELPP_LITERAL	303
9.3.1.182 ELPP_LOGGING_ENABLED	303
9.3.1.183 ELPP_MIN_UNIT	304
9.3.1.184 ELPP_MINGW	304
9.3.1.185 ELPP_OS_AIX	304
9.3.1.186 ELPP_OS_ANDROID	304
9.3.1.187 ELPP_OS_EMSCRIPTEN	304
9.3.1.188 ELPP_OS_FREEBSD	304
9.3.1.189 ELPP_OS_LINUX	304
9.3.1.190 ELPP_OS_MAC	304
9.3.1.191 ELPP_OS_NETBSD	305
9.3.1.192 ELPP_OS_QNX	305
9.3.1.193 ELPP_OS_SOLARIS	305
9.3.1.194 ELPP_OS_UNIX	305
9.3.1.195 ELPP_OS_WINDOWS	305

9.3.1.196 ELPP_SIMPLE_LOG	305
9.3.1.197 ELPP_STACKTRACE	305
9.3.1.198 ELPP_STRLLEN	306
9.3.1.199 ELPP_THREADING_ENABLED	306
9.3.1.200 ELPP_TRACE	306
9.3.1.201 ELPP_TRACE_LOG	306
9.3.1.202 ELPP_UNUSED	306
9.3.1.203 ELPP_USE_DEF_CRASH_HANDLER	306
9.3.1.204 ELPP_USE_STD_THREADING	306
9.3.1.205 ELPP_VARIADIC_TEMPLATES_SUPPORTED	307
9.3.1.206 ELPP_VERBOSE_LOG	307
9.3.1.207 ELPP_WARNING_LOG	307
9.3.1.208 ELPP_WRITE_LOG	307
9.3.1.209 ELPP_WRITE_LOG_AFTER_N	307
9.3.1.210 ELPP_WRITE_LOG_EVERY_N	308
9.3.1.211 ELPP_WRITE_LOG_IF	308
9.3.1.212 ELPP_WRITE_LOG_N_TIMES	308
9.3.1.213 ELPP_WX_ENABLED	308
9.3.1.214 ELPP_WX_HASH_MAP_ENABLED	309
9.3.1.215 ELPP_WX_PTR_ENABLED	309
9.3.1.216 elptime	309
9.3.1.217 elptime_r	309
9.3.1.218 elptime_s	309
9.3.1.219 INITIALIZE_EASYLOGGINGPP	309
9.3.1.220 INITIALIZE_NULL_EASYLOGGINGPP	309
9.3.1.221 LOG	310
9.3.1.222 LOG_AFTER_N	310
9.3.1.223 LOG_EVERY_N	310
9.3.1.224 LOG_IF	310
9.3.1.225 LOG_N_TIMES	310
9.3.1.226 MAKE_CONTAINERELPP_FRIENDLY	311
9.3.1.227 MAKE_LOGGABLE	311
9.3.1.228 PCHECK	311
9.3.1.229 PERFORMANCE_CHECKPOINT	312
9.3.1.230 PERFORMANCE_CHECKPOINT_WITH_ID	312
9.3.1.231 PLOG	312
9.3.1.232 PLOG_IF	312
9.3.1.233 SHARE_EASYLOGGINGPP	312
9.3.1.234 START_EASYLOGGINGPP	312
9.3.1.235 STRCAT	313
9.3.1.236 STRCPY	313
9.3.1.237 STRError	313

9.3.1.238 STRTOK	313
9.3.1.239 SYSLOG	313
9.3.1.240 SYSLOG_AFTER_N	313
9.3.1.241 SYSLOG_EVERY_N	314
9.3.1.242 SYSLOG_IF	314
9.3.1.243 SYSLOG_N_TIMES	314
9.3.1.244 TIMED_BLOCK	314
9.3.1.245 TIMED_FUNC	314
9.3.1.246 TIMED_FUNC_IF	315
9.3.1.247 TIMED_SCOPE	315
9.3.1.248 TIMED_SCOPE_IF	315
9.3.1.249 VLOG	315
9.3.1.250 VLOG_AFTER_N	316
9.3.1.251 VLOG_EVERY_N	316
9.3.1.252 VLOG_IF	316
9.3.1.253 VLOG_IS_ON	316
9.3.1.254 VLOG_N_TIMES	316
9.4 easylogging++.h	317
9.5 README.md File Reference	365
9.6 src/main.cpp File Reference	365
9.6.1 Function Documentation	365
9.6.1.1 main()	365
9.7 main.cpp	366
Index	367

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
 - * ubuntu 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:

el	Easylogging++ entry namespace	15
el::base	Namespace containing base/internal functionality used by Easylogging++	20
el::base::consts	Namespace containing constants used internally	23
el::base::debug	Contains some internal debugging tools like crash handler and stack tracer	35
el::base::threading	35
el::base::threading::internal	36
el::base::type	Data types used by Easylogging++	37
el::base::utils	Namespace containing utility functions/static classes used internally	39
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	41
std	42
WIP	Namespace for work in progress	42

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

el::base::utils::CommandLineArgs	53
el::ConfigurationStringToTypeItem	71
el::base::debug::CrashHandler	74
el::CustomFormatSpecifier	75
std::hash< el::Level >	85
el::base::HitCounter	91
el::LogDispatchData	102
el::Loggable	112
el::Configuration	57
el::Logger	113
el::base::LogFormat	106
el::LogMessage	133
el::base::MessageBuilder	136
el::base::NoCopy	138
el::LogBuilder	97
el::base::DefaultLogBuilder	79
el::base::LogDispatcher	104
el::base::NullWriter	142
el::base::Storage	177
el::base::VRegistry	211
el::base::Writer	215
el::base::PErrorWriter	150
el::base::threading::internal::NoMutex	139
el::base::threading::internal::NoScopedLock< Mutex >	141
el::base::HitCounter::Predicate	152
el::Configuration::Predicate	153
el::Loggers::ScopedAddFlag	174
el::Loggers::ScopedRemoveFlag	175
el::base::StaticClass	176
el::ConfigurationTypeHelper	72
el::Configurations::Parser	146
el::Helpers	86
el::LevelHelper	95
el::Loggers	124

el::VersionInfo	210
el::base::utils::DateTime	77
el::base::utils::File	83
el::base::utils::OS	144
el::base::utils::Str	187
el::StringToLevelItem	193
el::base::SubsecondPrecision	194
el::SysLogInitializer	195
el::base::threading::ThreadSafe	197
el::Callback< LogDispatchData >	51
el::LogDispatchCallback	100
el::base::DefaultLogDispatchCallback	81
el::Callback< Logger >	51
el::LoggerRegistrationCallback	123
el::Callback< PerformanceTrackingData >	51
el::PerformanceTrackingCallback	149
el::base::utils::AbstractRegistry< Configuration, std::vector< Configuration * > >	45
el::base::utils::AbstractRegistry< base::HitCounter, std::vector< base::HitCounter * > >	45
el::base::utils::AbstractRegistry< Logger, std::unordered_map< std::string, Logger * > >	45
el::base::utils::AbstractRegistry< T_Ptr, std::unordered_map< const char *, T_Ptr * > >	45
el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr * > >	45
el::base::utils::RegistryWithPred< T_Ptr, Pred >	169
el::Callback< T >	51
el::Logger	113
el::base::Storage	177
el::base::TypedConfigurations	199
el::base::VRegistry	211
el::base::utils::AbstractRegistry< T_Ptr, Container >	45
el::base::utils::Registry< Logger, std::string >	165
el::base::RegisteredLoggers	158
el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >	169
el::Configurations	60
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >	169
el::base::RegisteredHitCounters	155
el::base::utils::Registry< T_Ptr, T_Key >	165
el::base::utils::Utils	209

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

el::base::utils::AbstractRegistry< T_Ptr, Container >	
Abstract registry (aka repository) that provides basic interface for pointer repository specified by T_Ptr type	45
el::Callback< T >	51
el::base::utils::CommandLineArgs	
Command line arguments for application if specified using el::Helpers::setArgs(..) or START_↔ EASYLOGGINGPP(..)	53
el::Configuration	
Represents single configuration that has representing level, configuration type and a string based value	57
el::Configurations	
Thread-safe Configuration repository	60
el::ConfigurationStringToTypeItem	71
el::ConfigurationTypeHelper	
Static class that contains helper functions for el::ConfigurationType	72
el::base::debug::CrashHandler	74
el::CustomFormatSpecifier	
User-provided custom format specifier	75
el::base::utils::DateTime	
Contains utilities for cross-platform date/time. This class make use of el::base::utils::Str	77
el::base::DefaultLogBuilder	79
el::base::DefaultLogDispatchCallback	81
el::base::utils::File	83
std::hash< el::Level >	85
el::Helpers	
Static helpers for developers	86
el::base::HitCounter	
Class that keeps record of current line hit for occasional logging	91
el::LevelHelper	
Static class that contains helper functions for el::Level	95
el::LogBuilder	97
el::LogDispatchCallback	100
el::LogDispatchData	102
el::base::LogDispatcher	
Dispatches log messages	104

el::base::LogFormat	Represents log format containing flags and date format. This is used internally to start initial log	106
el::Loggable	Base of Easylogging++ friendly class	112
el::Logger	Represents a logger holding ID and configurations we need to write logs	113
el::LoggerRegistrationCallback		123
el::Loggers	Static helpers to deal with loggers and their configurations	124
el::LogMessage		133
el::base::MessageBuilder		136
el::base::NoCopy	Internal helper class that prevent copy constructor for class	138
el::base::threading::internal::NoMutex	Mutex wrapper used when multi-threading is disabled	139
el::base::threading::internal::NoScopedLock< Mutex >	Lock guard wrapper used when multi-threading is disabled	141
el::base::NullWriter	Writes nothing - Used when certain log is disabled	142
el::base::utils::OS	Operating System helper static class used internally. You should not use it	144
el::Configurations::Parser	Parser used internally to parse configurations from file or text	146
el::PerformanceTrackingCallback		149
el::base::PErrorWriter		150
el::base::HitCounter::Predicate		152
el::Configuration::Predicate	Used to find configuration from configuration (pointers) repository. Avoid using it	153
el::base::RegisteredHitCounters	Repository for hit counters used across the application	155
el::base::RegisteredLoggers	Loggers repository	158
el::base::utils::Registry< T_Ptr, T_Key >	A pointer registry mechanism to manage memory and provide search functionalities. (non-predicate version)	165
el::base::utils::RegistryWithPred< T_Ptr, Pred >	A pointer registry mechanism to manage memory and provide search functionalities. (predicate version)	169
el::Loggers::ScopedAddFlag	Adds flag and removes it when scope goes out	174
el::Loggers::ScopedRemoveFlag	Removes flag and add it when scope goes out	175
el::base::StaticClass	Internal helper class that makes all default constructors private	176
el::base::Storage	Easylogging++ management storage	177
el::base::utils::Str	String utilities helper class used internally. You should not use it	187
el::StringToLevelItem		193
el::base::SubsecondPrecision	A subsecond precision class containing actual width and offset of the subsecond part	194
el::SysLogInitializer	Initializes syslog with process ID, options and facility. calls closelog() on d'tor	195
el::base::threading::ThreadSafe	Base of thread safe class, this class is inheritable-only	197
el::base::TypedConfigurations	Configurations with data types	199
el::base::utils::Utils		209

el::VersionInfo	210
el::base::VRegistry Represents registries for verbose logging	211
el::base::Writer Main entry point of each logging	215

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

lib/ easylogging++.cc	221
lib/ easylogging++.h	260
src/ main.cpp	365

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 [ConfigurationStringToTypeltem](#)

- 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 terminal, 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.
AllowVerboselfModuleNotSpecified	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::vall, 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\(\)](#).

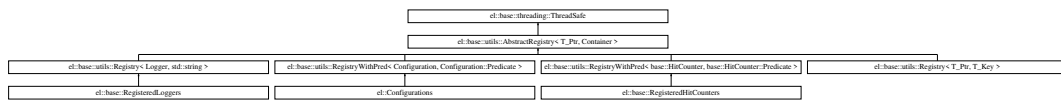
Chapter 8

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 >`:



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](#).

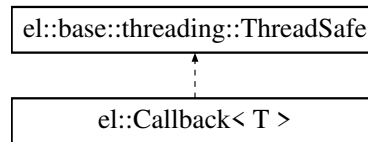
The documentation for this class was generated from the following file:

- [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:

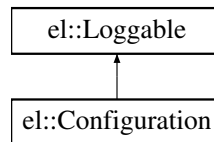
- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

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

<i>value</i>	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:

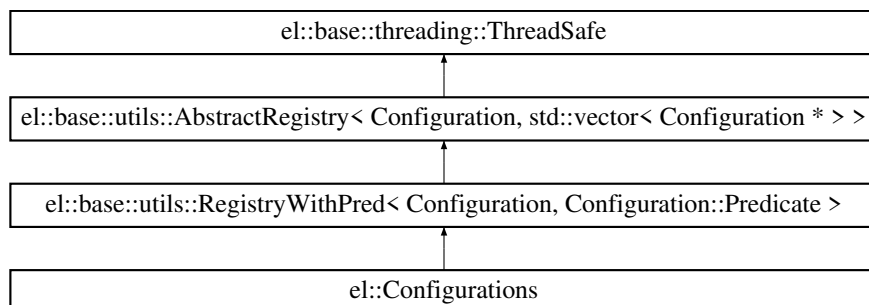
- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

8.5 el::Configurations Class Reference

Thread-safe [Configuration](#) repository.

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

Inheritance diagram for el::Configurations:



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

<i>configurationFile</i>	Full path to configuration file
<i>useDefaultsForRemaining</i>	Lets you set the remaining configurations to default.
<i>base</i>	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

<i>configurationType</i>	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

<i>level</i>	Level to check
<i>configurationType</i>	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

<i>configurationFile</i>	Full path to configuration file
<i>base</i>	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

<i>base</i>	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

<i>level</i>	Level to set configuration for (el::Level).
<i>configurationType</i>	Type of configuration (el::ConfigurationType)
<i>value</i>	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::Predicate >::RegistryWithPred\(\)](#), [unsafeSet\(\)](#), and [unsafeSetGlobally\(\)](#).

8.5.3.10 setFromBase()

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

Sets configuration based-off an existing configurations.

Parameters

<i>base</i>	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

<i>configurationType</i>	Type of configuration
<i>value</i>	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< Configuration](#), [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:

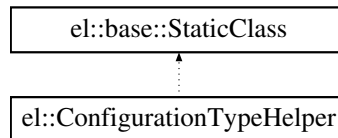
- [lib/easylogging++.cc](#)

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

<i>configStr</i>	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

<i>startIndex</i>	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.
<i>fn</i>	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](#).

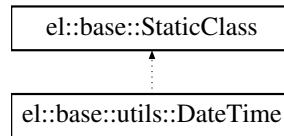
The documentation for this class was generated from the following file:

- [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

<i>format</i>	User provided date/time format
<i>ssPrec</i>	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](#).

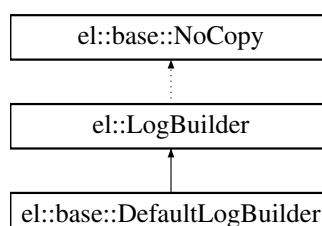
References [buildTimeInfo\(\)](#), and [parseFormat\(\)](#).

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](#).

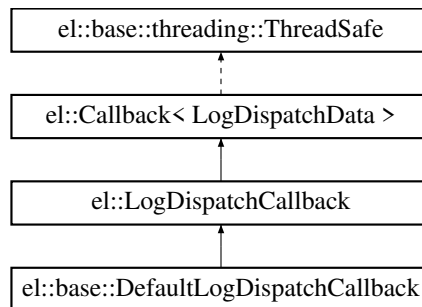
References [el::base::utils::Str::addToBuff\(\)](#), [el::base::AppName](#), [el::base::utils::File::buildBaseFilename\(\)](#), [el::base::utils::File::buildStrippedFilename\(\)](#), [el::base::utils::Str::clearBuff\(\)](#), [el::base::utils::Str::convertAndAddToBuff\(\)](#), [el::base::DateTime](#), [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::kLogFileBaseFormatSpecifier](#), [el::base::consts::kLogFileFormatSpecifier](#), [el::base::consts::kLogFunctionFormatSpecifier](#), [el::base::consts::kLogLineFormatSpecifier](#), [el::base::consts::kLogLocationFormatSpecifier](#), [el::base::consts::kMessageFormatSpecifier](#), [el::base::consts::kSourceFilenameMaxLength](#), [el::base::consts::kSourceLineMaxLength](#), [el::base::consts::kThreadIdFormatSpecifier](#), [el::base::consts::kVerboseLevelFormatSpecifier](#), [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:



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]
```

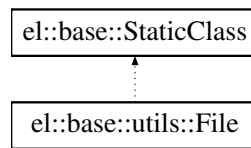
Definition at line 2730 of file [easylogging++.h](#).

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::kSourceFilenameMaxLength, 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

<i>path</i>	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:

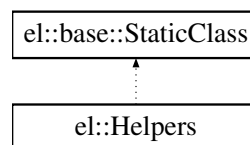
- [lib/easylogging++.h](#)

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:

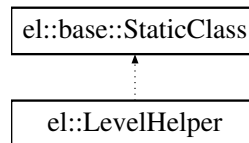
- [lib/easylogging++.h](#)

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

<i>levelStr</i>	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

<i>startIndex</i>	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.
<i>fn</i>	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](#).

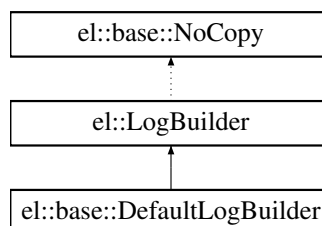
The documentation for this class was generated from the following files:

- [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](#).

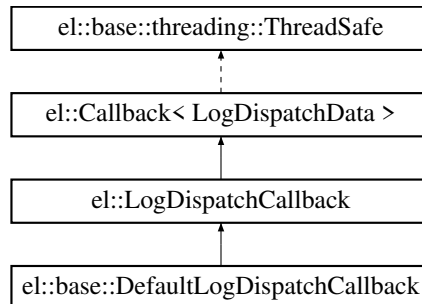
The documentation for this class was generated from the following files:

- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

8.19 el::LogDispatchCallback Class Reference

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

Inheritance diagram for el::LogDispatchCallback:



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](#).

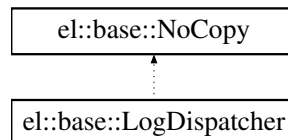
The documentation for this class was generated from the following file:

- [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](#).

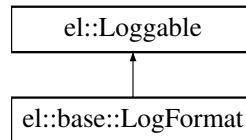
The documentation for this class was generated from the following files:

- [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

<i>userFormat</i>	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::kLogFunctionFormatSpecifier](#), [el::base::consts::kLoggerIdFormatSpecifier](#), [el::base::consts::kLogLineFormatSpecifier](#), [el::base::consts::kLogLocationFormatSpecifier](#), [el::base::consts::kMessageFormatSpecifier](#), [el::base::consts::kSeverityLevelFormatSpecifier](#), [el::base::consts::kSeverityLevelShortFormatSpecifier](#), [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

	<i>index</i>	Index where datetime, date or time was found
<i>in, out</i>	<i>currFormat</i>	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::kTraceLevelLogValue](#), [el::base::consts::kTraceLevelShortLogValue](#), [el::base::consts::kVerboseLevelLogValue](#), [el::base::consts::kVerboseLevelShortLogValue](#), [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:

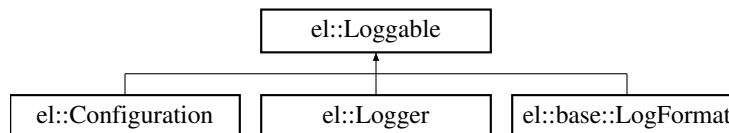
- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

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:

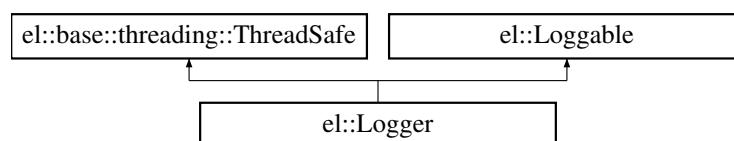
- [lib/easylogging++.h](#)

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](#).

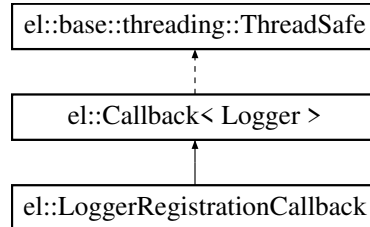
The documentation for this class was generated from the following files:

- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

8.25 el::LoggerRegistrationCallback Class Reference

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

Inheritance diagram for el::LoggerRegistrationCallback:



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:

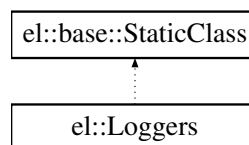
- lib/[easylogging++.h](#)

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	<i>targetList</i>	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 writeliterator()

```
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:

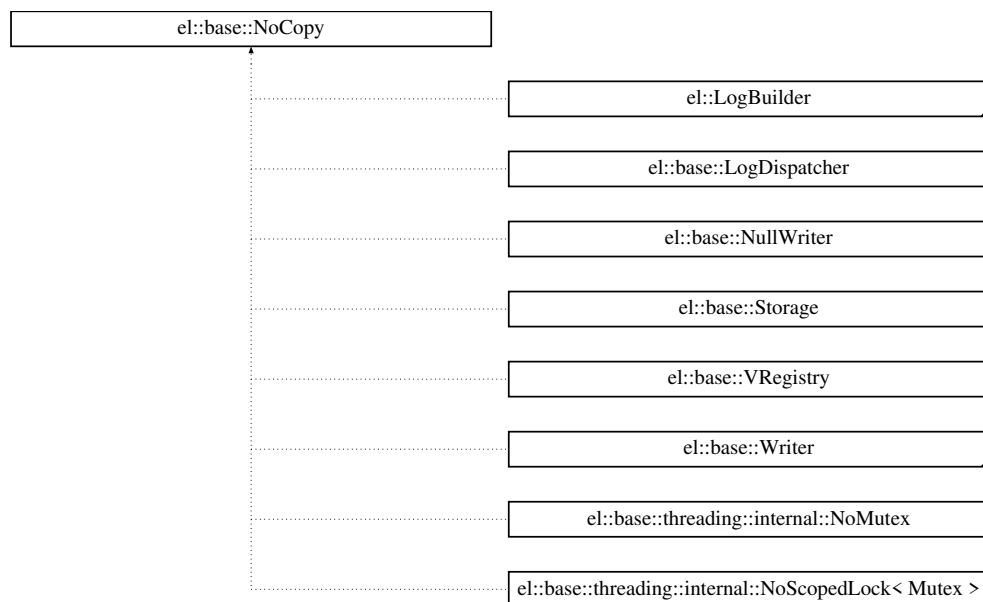
- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

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]
```

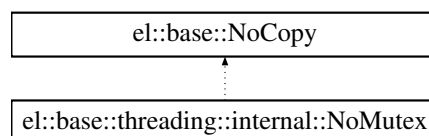
The documentation for this class was generated from the following file:

- 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](#).

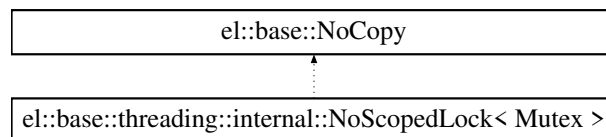
The documentation for this class was generated from the following file:

- 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]
```

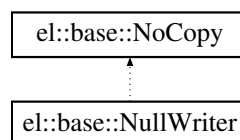
The documentation for this class was generated from the following file:

- [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](#).

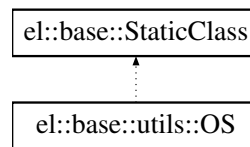
The documentation for this class was generated from the following file:

- 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

<i>command</i>	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

<i>variableName</i>	Environment variable name
<i>defaultVal</i>	If no environment variable or value found the value to return by default
<i>alternativeBashCommand</i>	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:

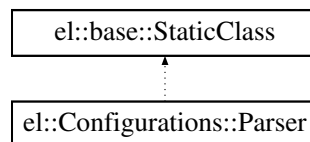
- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

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

<i>configurationFile</i>	Full path to configuration file
<i>sender</i>	Sender configurations pointer. Usually 'this' is used from calling class
<i>base</i>	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, Configuration::Predicate >::RegistryWithPred\(\)](#), 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

<i>configurationsString</i>	the configuration in plain text format
<i>sender</i>	Sender configurations pointer. Usually 'this' is used from calling class
<i>base</i>	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](#).

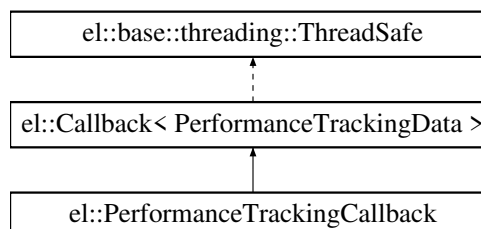
The documentation for this class was generated from the following files:

- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

8.35 el::PerformanceTrackingCallback Class Reference

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

Inheritance diagram for `el::PerformanceTrackingCallback`:



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]
```

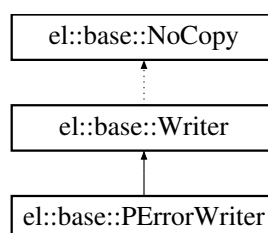
Definition at line [2191](#) of file [easylogging++.h](#).

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](#).

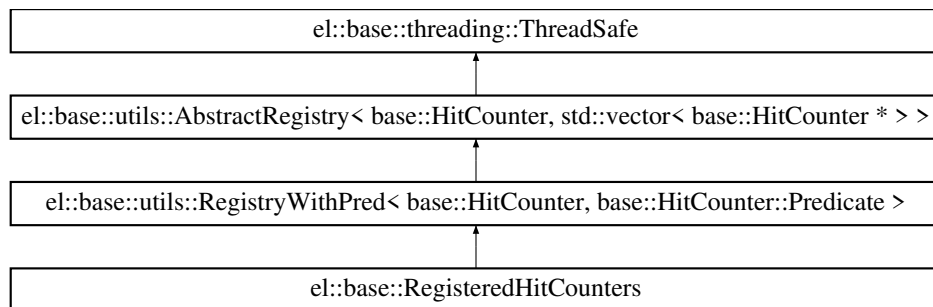
The documentation for this class was generated from the following files:

- [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:



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 $\geq N$, 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 $\leq n$, 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 $\geq N$, 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::ThreadSafe](#), [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::ThreadSafe](#), [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 $\leq n$, 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::ThreadSafe](#), [el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::registerNew\(\)](#), and [el::base::utils::RegistryWithP](#)

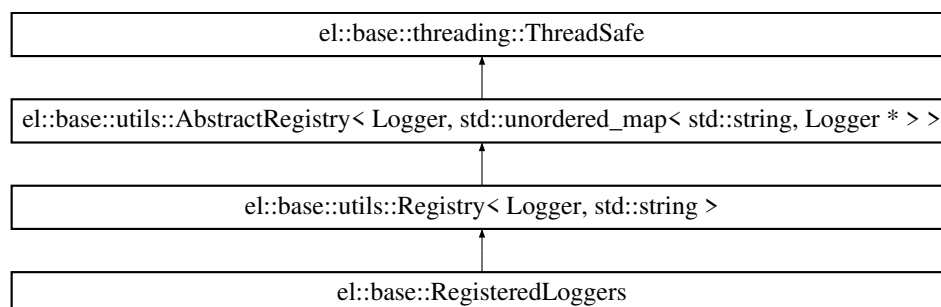
The documentation for this class was generated from the following files:

- [lib/easylogging++.h](#)
- [lib/easylogging++.cc](#)

8.40 el::base::RegisteredLoggers Class Reference

[Loggers](#) repository.

Inheritance diagram for `el::base::RegisteredLoggers`:



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::setDefault\(\)](#).

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::RegisteredLoggers::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](#).

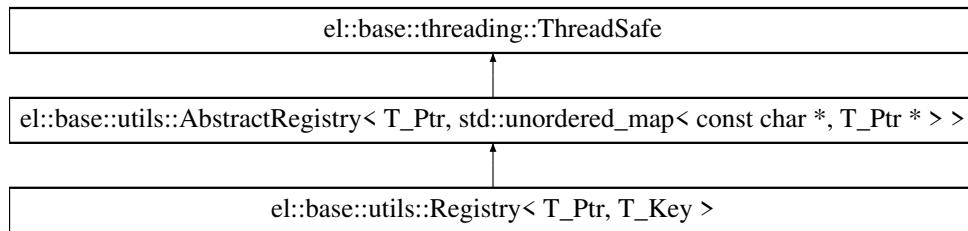
The documentation for this class was generated from the following files:

- [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 >:



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](#).

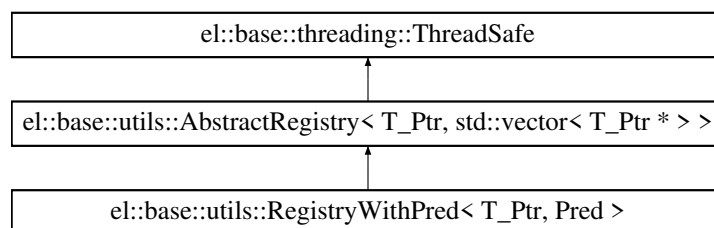
The documentation for this class was generated from the following file:

- [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 >](#):



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:

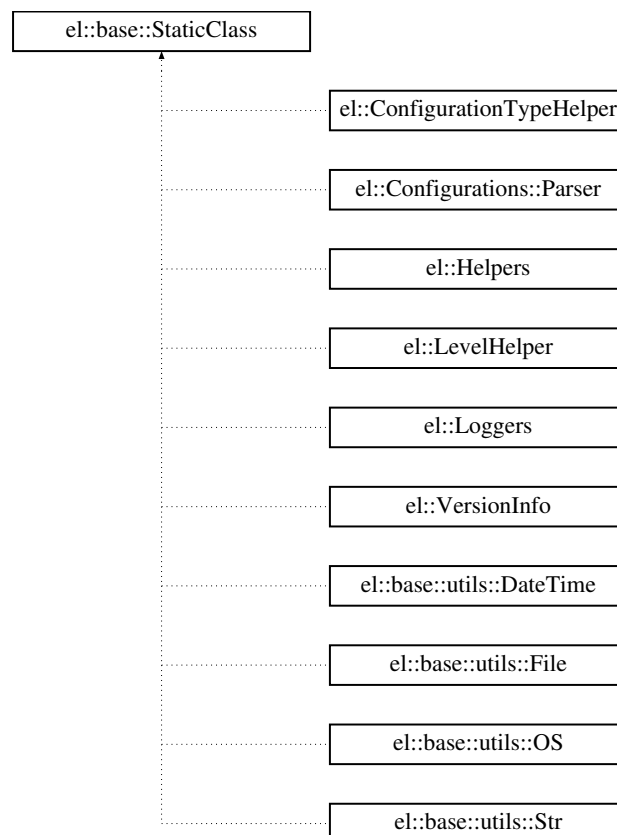
- [lib/easylogging++.h](#)

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]
```

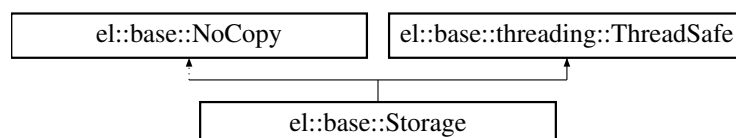
The documentation for this class was generated from the following file:

- 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_logDispatchCallbacks [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](#).

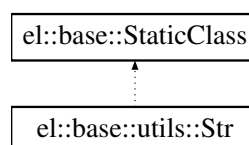
The documentation for this class was generated from the following files:

- [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

<i>str</i>	String to check
<i>end</i>	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

<i>in, out</i>	<i>str</i>	String to replace from
	<i>replaceWhat</i>	Character to replace
	<i>replaceWith</i>	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 >::end\(\)](#).

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

<i>str</i>	String to replace from
<i>replaceWhat</i>	Character to replace
<i>replaceWith</i>	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

<i>str</i>	String to check
<i>start</i>	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

<i>str</i>	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 >::end\(\)](#).

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](#).

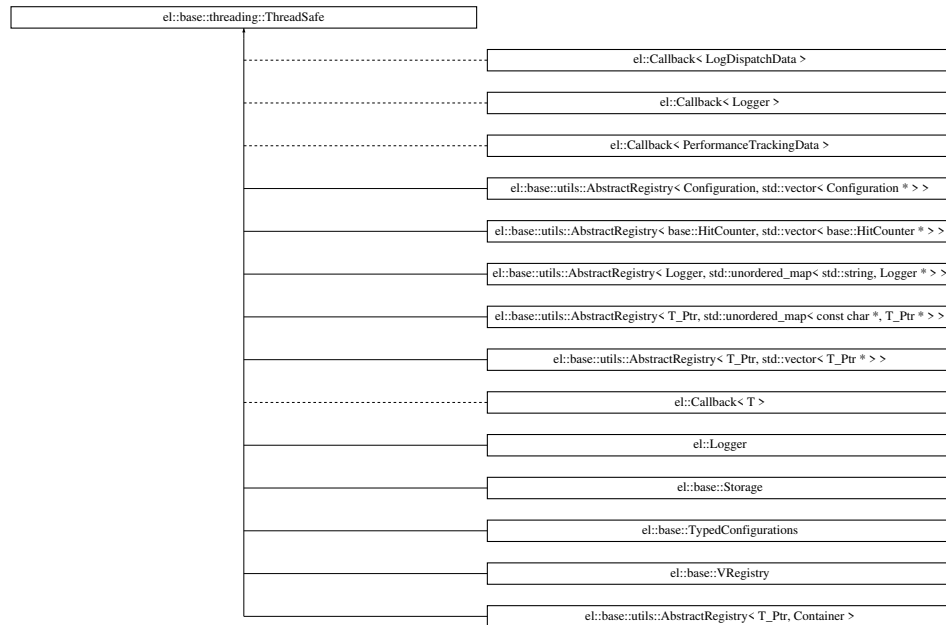
The documentation for this class was generated from the following file:

- 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:



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](#).

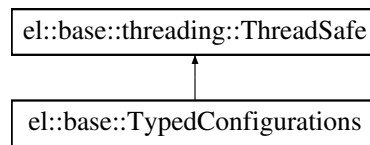
The documentation for this class was generated from the following file:

- [lib/easylogging++.h](#)

8.52 el::base::TypedConfigurations Class Reference

[Configurations](#) with data types.

Inheritance diagram for el::base::TypedConfigurations:



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 *confName)
- 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 what's 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

<i>configurations</i>	Configurations pointer/reference to base this typed configurations off.
<i>logStreamsReference</i>	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::getFileSize\(\)](#), [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_subsecondPrecisionMap [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\(\)](#).

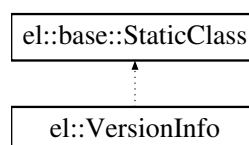
The documentation for this class was generated from the following file:

- [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](#).

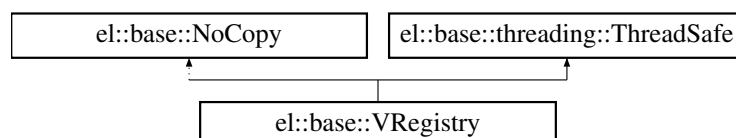
The documentation for this class was generated from the following files:

- [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](#).

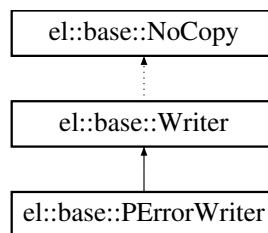
The documentation for this class was generated from the following files:

- [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.](#)

```
00001 //
00002 // Bismillah ar-Rahmaan ar-Raheem
00003 //
00004 // Easylogging++ v9.97.1
00005 // Cross-platform logging library for C++ applications
00006 //
00007 // Copyright (c) 2012-present @abumq (Majid Q.)
00008 //
00009 // This library is released under the MIT Licence.
00010 // https://github.com/amrayn/easyloggingpp/blob/master/LICENSE
00011 //
00012
00013 #include "easylogging++.h"
00014
00015 #if defined(AUTO_INITIALIZE_EASYLOGGINGPP)
00016 INITIALIZE_EASYLOGGINGPP
00017 #endif
00018
00019 namespace el {
00020
00021 // el::base
00022 namespace base {
00023 // el::base::consts
00024 namespace consts {
00025
00026 // Level log values - These are values that are replaced in place of %level format specifier
00027 // Extra spaces after format specifiers are only for readability purposes in log files
00028 static const base::type::char_t* kInfoLevelLogValue = ELPP_LITERAL("INFO");
00029 static const base::type::char_t* kDebugLevelLogValue = ELPP_LITERAL("DEBUG");
00030 static const base::type::char_t* kWarningLevelLogValue = ELPP_LITERAL("WARNING");
00031 static const base::type::char_t* kErrorLevelLogValue = ELPP_LITERAL("ERROR");
00032 static const base::type::char_t* kFatalLevelLogValue = ELPP_LITERAL("FATAL");
00033 static const base::type::char_t* kVerboseLevelLogValue =
00034     ELPP_LITERAL("VERBOSE"); // will become VERBOSE-x where x = verbose level
00035 static const base::type::char_t* kTraceLevelLogValue = ELPP_LITERAL("TRACE");
00036 static const base::type::char_t* kInfoLevelShortLogValue = ELPP_LITERAL("I");
00037 static const base::type::char_t* kDebugLevelShortLogValue = ELPP_LITERAL("D");
00038 static const base::type::char_t* kWarningLevelShortLogValue = ELPP_LITERAL("W");
00039 static const base::type::char_t* kErrorLevelShortLogValue = ELPP_LITERAL("E");
00040 static const base::type::char_t* kFatalLevelShortLogValue = ELPP_LITERAL("F");
00041 static const base::type::char_t* kVerboseLevelShortLogValue = ELPP_LITERAL("V");
00042 static const base::type::char_t* kTraceLevelShortLogValue = ELPP_LITERAL("T");
00043 // Format specifiers - These are used to define log format
00044 static const base::type::char_t* kAppNameFormatSpecifier = ELPP_LITERAL("%app");
00045 static const base::type::char_t* kLoggerIdFormatSpecifier = ELPP_LITERAL("%logger");
00046 static const base::type::char_t* kThreadIdFormatSpecifier = ELPP_LITERAL("%thread");
00047 static const base::type::char_t* kSeverityLevelFormatSpecifier = ELPP_LITERAL("%level");
00048 static const base::type::char_t* kSeverityLevelShortFormatSpecifier =
00049     ELPP_LITERAL("%levshort");
00049 static const base::type::char_t* kDateTimeFormatSpecifier = ELPP_LITERAL("%datetime");
00050 static const base::type::char_t* kLogFileFormatSpecifier = ELPP_LITERAL("%file");
00051 static const base::type::char_t* kLogFileNameFormatSpecifier = ELPP_LITERAL("%fbase");
00052 static const base::type::char_t* kLogLineFormatSpecifier = ELPP_LITERAL("%line");
00053 static const base::type::char_t* kLogLocationFormatSpecifier = ELPP_LITERAL("%loc");
00054 static const base::type::char_t* kLogFunctionFormatSpecifier = ELPP_LITERAL("%func");
00055 static const base::type::char_t* kCurrentUserFormatSpecifier = ELPP_LITERAL("%user");
00056 static const base::type::char_t* kCurrentHostFormatSpecifier = ELPP_LITERAL("%host");
00057 static const base::type::char_t* kMessageFormatSpecifier = ELPP_LITERAL("%msg");
00058 static const base::type::char_t* kVerboseLevelFormatSpecifier = ELPP_LITERAL("%vlevel");
00059 static const char* kDateTimeFormatSpecifierForFilename = "%datetime";
00060 // Date/time
00061 static const char* kDays[7] = { "Sunday", "Monday", "Tuesday",
00062     "Wednesday", "Thursday", "Friday", "Saturday" };
00062 static const char* kDaysAbbrev[7] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
00063     "Sat" };

```

```

00063 static const char* kMonths[12] = { "January", "February", "March", "April",
00064   "May", "June", "July", "August",
    "September", "October", "November",
00065   "December" };
00066 static const char* kMonthsAbbrev[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
00067 static const char* kDefaultDateTimeFormat = "%Y-%M-%d %H:%m:%s,%g";
00068 static const char* kDefaultDateTimeFormatInFilename = "%Y-%M-%d_%H-%m";
00069 static const int kYearBase = 1900;
00070 static const char* kAm = "AM";
00071 static const char* kPm = "PM";
00072 // Miscellaneous constants
00073
00074 static const char* kNullPointer = "nullptr";
00075 #if ELPP_VARIADIC_TEMPLATES_SUPPORTED
00076 #endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
00077 static const base::type::VerboseLevel kMaxVerboseLevel = 9;
00078 static const char* kUnknownUser = "unknown-user";
00079 static const char* kUnknownHost = "unknown-host";
00080
00081
00082 //----- DEFAULT LOG FILE -----
00083
00084 #if defined(ELPP_NO_DEFAULT_LOG_FILE)
00085 #   if ELPP_OS_UNIX
00086 static const char* kDefaultLogFile = "/dev/null";
00087 #   elif ELPP_OS_WINDOWS
00088 static const char* kDefaultLogFile = "nul";
00089 #   endif // ELPP_OS_UNIX
00090 #elif defined(ELPP_DEFAULT_LOG_FILE)
00091 static const char* kDefaultLogFile = ELPP_DEFAULT_LOG_FILE;
00092 #else
00093 static const char* kDefaultLogFile = "myeasylog.log";
00094 #endif // defined(ELPP_NO_DEFAULT_LOG_FILE)
00095
00096
00097 #if !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
00098 static const char* kDefaultLogFileParam = "--default-log-file";
00099 #endif // !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
00100 #if defined(ELPP_LOGGING_FLAGS_FROM_ARG)
00101 static const char* kLoggingFlagsParam = "--logging-flags";
00102 #endif // defined(ELPP_LOGGING_FLAGS_FROM_ARG)
00103 static const char* kValidLoggerIdSymbols =
00104   "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._";
00105 static const char* kConfigurationComment = "##";
00106 static const char* kConfigurationLevel = "*";
00107 static const char* kConfigurationLoggerId = "--";
00108 }
00109 // el::base::utils
00110 namespace utils {
00111
00112 static void abort(int status, const std::string& reason) {
00113   // Both status and reason params are there for debugging with tools like gdb etc
00114   ELPP_UNUSED(status);
00115   ELPP_UNUSED(reason);
00116   #if defined(ELPP_COMPILER_MSVC) && defined(_M_IX86) && defined(_DEBUG)
00117     // Ignore msvc critical error dialog - break instead (on debug mode)
00118     _asm int 3
00119   #else
00120     ::abort();
00121   #endif // defined(ELPP_COMPILER_MSVC) && defined(_M_IX86) && defined(_DEBUG)
00122 }
00123
00124 } // namespace utils
00125 } // namespace base
00126
00127 // el
00128
00129 // LevelHelper
00130
00131 const char* LevelHelper::convertToString(Level level) {
00132   // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
00133   if (level == Level::Global) return "GLOBAL";
00134   if (level == Level::Debug) return "DEBUG";
00135   if (level == Level::Info) return "INFO";
00136   if (level == Level::Warning) return "WARNING";
00137   if (level == Level::Error) return "ERROR";
00138   if (level == Level::Fatal) return "FATAL";
00139   if (level == Level::Verbose) return "VERBOSE";
00140   if (level == Level::Trace) return "TRACE";
00141   return "UNKNOWN";
00142 }
00143
00144 struct StringToLevelItem {
00145   const char* levelString;
00146   Level level;

```



```

00148 };
00149
00150 static struct StringToLevelItem stringToLevelMap[] = {
00151     { "global", Level::Global },
00152     { "debug", Level::Debug },
00153     { "info", Level::Info },
00154     { "warning", Level::Warning },
00155     { "error", Level::Error },
00156     { "fatal", Level::Fatal },
00157     { "verbose", Level::Verbose },
00158     { "trace", Level::Trace }
00159 };
00160
00161 Level LevelHelper::convertFromString(const char* levelStr) {
00162     for (auto& item : stringToLevelMap) {
00163         if (base::utils::Str::cStringCaseEq(levelStr, item.levelString)) {
00164             return item.level;
00165         }
00166     }
00167     return Level::Unknown;
00168 }
00169
00170 void LevelHelper::forEachLevel(base::type::EnumType* startIndex, const std::function<bool(void)>& fn)
00171 {
00172     base::type::EnumType lIndexMax = LevelHelper::kMaxValid;
00173     do {
00174         if (fn()) {
00175             break;
00176         }
00177         *startIndex = static_cast<base::type::EnumType>(*startIndex << 1);
00178     } while (*startIndex <= lIndexMax);
00179 }
00180 // ConfigurationTypeHelper
00181
00182 const char* ConfigurationTypeHelper::convertToString(ConfigurationType configurationType) {
00183     // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
00184     if (configurationType == ConfigurationType::Enabled) return "ENABLED";
00185     if (configurationType == ConfigurationType::Filename) return "FILENAME";
00186     if (configurationType == ConfigurationType::Format) return "FORMAT";
00187     if (configurationType == ConfigurationType::ToFile) return "TO_FILE";
00188     if (configurationType == ConfigurationType::ToStandardOutput) return "TO_STANDARD_OUTPUT";
00189     if (configurationType == ConfigurationType::SubsecondPrecision) return "SUBSECOND_PRECISION";
00190     if (configurationType == ConfigurationType::PerformanceTracking) return "PERFORMANCE_TRACKING";
00191     if (configurationType == ConfigurationType::MaxLogFileSize) return "MAX_LOG_FILE_SIZE";
00192     if (configurationType == ConfigurationType::LogFlushThreshold) return "LOG_FLUSH_THRESHOLD";
00193     return "UNKNOWN";
00194 }
00195
00196 struct ConfigurationStringToTypeItem {
00197     const char* configString;
00198     ConfigurationType configType;
00199 };
00200
00201 static struct ConfigurationStringToTypeItem configStringToTypeMap[] = {
00202     { "enabled", ConfigurationType::Enabled },
00203     { "to_file", ConfigurationType::ToFile },
00204     { "to_standard_output", ConfigurationType::ToStandardOutput },
00205     { "format", ConfigurationType::Format },
00206     { "filename", ConfigurationType::Filename },
00207     { "subsecond_precision", ConfigurationType::SubsecondPrecision },
00208     { "milliseconds_width", ConfigurationType::MillisecondsWidth },
00209     { "performance_tracking", ConfigurationType::PerformanceTracking },
00210     { "max_log_file_size", ConfigurationType::MaxLogFileSize },
00211     { "log_flush_threshold", ConfigurationType::LogFlushThreshold },
00212 };
00213
00214 ConfigurationType ConfigurationTypeHelper::convertFromString(const char* configStr) {
00215     for (auto& item : configStringToTypeMap) {
00216         if (base::utils::Str::cStringCaseEq(configStr, item.configString)) {
00217             return item.configType;
00218         }
00219     }
00220     return ConfigurationType::Unknown;
00221 }
00222
00223 void ConfigurationTypeHelper::forEachConfigType(base::type::EnumType* startIndex, const
std::function<bool(void)>& fn) {
00224     base::type::EnumType cIndexMax = ConfigurationTypeHelper::kMaxValid;
00225     do {
00226         if (fn()) {
00227             break;
00228         }
00229         *startIndex = static_cast<base::type::EnumType>(*startIndex << 1);
00230     } while (*startIndex <= cIndexMax);
00231 }
00232

```

```

00233 // Configuration
00234
00235 Configuration::Configuration(const Configuration& c) :
00236     m_level(c.m_level),
00237     m_configurationType(c.m_configurationType),
00238     m_value(c.m_value) {
00239 }
00240
00241 Configuration& Configuration::operator=(const Configuration& c) {
00242     if (&c != this) {
00243         m_level = c.m_level;
00244         m_configurationType = c.m_configurationType;
00245         m_value = c.m_value;
00246     }
00247     return *this;
00248 }
00249
00251 Configuration::Configuration(Level level, ConfigurationType configurationType, const std::string&
value) :
00252     m_level(level),
00253     m_configurationType(configurationType),
00254     m_value(value) {
00255 }
00256
00257 void Configuration::log(el::base::type::ostream_t& os) const {
00258     os << LevelHelper::convertToString(m_level)
00259     << ELPP_LITERAL(" ") << ConfigurationTypeHelper::convertToString(m_configurationType)
00260     << ELPP_LITERAL(" = ") << m_value.c_str();
00261 }
00262
00264 Configuration::Predicate::Predicate(Level level, ConfigurationType configurationType) :
00265     m_level(level),
00266     m_configurationType(configurationType) {
00267 }
00268
00269 bool Configuration::Predicate::operator()(const Configuration* conf) const {
00270     return ((conf != nullptr) && (conf->level() == m_level) && (conf->configurationType() ==
m_configurationType));
00271 }
00272
00273 // Configurations
00274
00275 Configurations::Configurations(void) :
00276     m_configurationFile(std::string()),
00277     m_isFromFile(false) {
00278 }
00279
00280 Configurations::Configurations(const std::string& configurationFile, bool useDefaultsForRemaining,
Configurations* base) :
00281     m_configurationFile(configurationFile),
00282     m_isFromFile(false) {
00283     parseFromFile(configurationFile, base);
00284     if (useDefaultsForRemaining) {
00285         setRemainingToDefault();
00286     }
00287 }
00288
00289
00290 bool Configurations::parseFromFile(const std::string& configurationFile, Configurations* base) {
00291     // We initial assertion with true because if we have assertion disabled, we want to pass this
00292     // check and if assertion is enabled we will have values re-assigned any way.
00293     bool assertionPassed = true;
00294     ELPP_ASSERT((assertionPassed = base::utils::File::pathExists(configurationFile.c_str(), true)) ==
true,
00295         "Configuration file [" << configurationFile << "] does not exist!");
00296     if (!assertionPassed) {
00297         return false;
00298     }
00299     bool success = Parser::parseFromFile(configurationFile, this, base);
00300     m_isFromFile = success;
00301     return success;
00302 }
00303
00304 bool Configurations::parseFromText(const std::string& configurationsString, Configurations* base) {
00305     bool success = Parser::parseFromText(configurationsString, this, base);
00306     if (success) {
00307         m_isFromFile = false;
00308     }
00309     return success;
00310 }
00311
00312 void Configurations::setFromBase(Configurations* base) {
00313     if (base == nullptr || base == this) {
00314         return;
00315     }
00316     base::threading::ScopedLock scopedLock(base->lock());
00317     for (Configuration*& conf : base->list()) {
00318         set(conf);

```

```

00319     }
00320 }
00321
00322 bool Configurations::hasConfiguration(ConfigurationType configurationType) {
00323     base::type::EnumType lIndex = LevelHelper::kMinValid;
00324     bool result = false;
00325     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
00326         if (hasConfiguration(LevelHelper::castFromInt(lIndex), configurationType)) {
00327             result = true;
00328         }
00329         return result;
00330     });
00331     return result;
00332 }
00333
00334 bool Configurations::hasConfiguration(Level level, ConfigurationType configurationType) {
00335     base::threading::ScopedLock scopedLock(lock());
00336     #if ELPP_COMPILER_INTEL
00337         // We cant specify template types here, Intel C++ throws compilation error
00338         // "error: type name is not allowed"
00339         return RegistryWithPred::get(level, configurationType) != nullptr;
00340     #else
00341         return RegistryWithPred<Configuration, Configuration::Predicate>::get(level, configurationType) !=
00342             nullptr;
00343     #endif // ELPP_COMPILER_INTEL
00344 }
00345 void Configurations::set(Level level, ConfigurationType configurationType, const std::string& value) {
00346     base::threading::ScopedLock scopedLock(lock());
00347     unsafeSet(level, configurationType, value); // This is not unsafe anymore as we have locked mutex
00348     if (level == Level::Global) {
00349         unsafeSetGlobally(configurationType, value, false); // Again this is not unsafe either
00350     }
00351 }
00352
00353 void Configurations::set(Configuration* conf) {
00354     if (conf == nullptr) {
00355         return;
00356     }
00357     set(conf->level(), conf->configurationType(), conf->value());
00358 }
00359
00360 void Configurations::setDefault(void) {
00361     setGlobally(ConfigurationType::Enabled, std::string("true"), true);
00362     setGlobally(ConfigurationType::Filename, std::string(base::consts::kDefaultLogFile), true);
00363     #if defined(ELPP_NO_LOG_TO_FILE)
00364     setGlobally(ConfigurationType::ToFile, std::string("false"), true);
00365     #else
00366     setGlobally(ConfigurationType::ToFile, std::string("true"), true);
00367     #endif // defined(ELPP_NO_LOG_TO_FILE)
00368     setGlobally(ConfigurationType::ToStandardOutput, std::string("true"), true);
00369     setGlobally(ConfigurationType::SubsecondPrecision, std::string("3"), true);
00370     setGlobally(ConfigurationType::PerformanceTracking, std::string("true"), true);
00371     setGlobally(ConfigurationType::MaxLogFileSize, std::string("0"), true);
00372     setGlobally(ConfigurationType::LogFlushThreshold, std::string("0"), true);
00373
00374     setGlobally(ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"), true);
00375     set(Level::Debug, ConfigurationType::Format,
00376         std::string("%datetime %level [%logger] [%user@%host] [%func] [%loc] %msg"));
00377     // INFO and WARNING are set to default by Level::Global
00378     set(Level::Error, ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"));
00379     set(Level::Fatal, ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"));
00380     set(Level::Verbose, ConfigurationType::Format, std::string("%datetime %level-%vlevel [%logger]
00381 %msg"));
00382     set(Level::Trace, ConfigurationType::Format, std::string("%datetime %level [%logger] [%func] [%loc]
00383 %msg"));
00384 }
00385
00386 void Configurations::setRemainingToDefault(void) {
00387     base::threading::ScopedLock scopedLock(lock());
00388     #if defined(ELPP_NO_LOG_TO_FILE)
00389     unsafeSetIfNotExist(Level::Global, ConfigurationType::Enabled, std::string("false"));
00390     #else
00391     unsafeSetIfNotExist(Level::Global, ConfigurationType::Enabled, std::string("true"));
00392     #endif // defined(ELPP_NO_LOG_TO_FILE)
00393     unsafeSetIfNotExist(Level::Global, ConfigurationType::Filename,
00394         std::string(base::consts::kDefaultLogFile));
00395     unsafeSetIfNotExist(Level::Global, ConfigurationType::ToStandardOutput, std::string("true"));
00396     unsafeSetIfNotExist(Level::Global, ConfigurationType::SubsecondPrecision, std::string("3"));
00397     unsafeSetIfNotExist(Level::Global, ConfigurationType::PerformanceTracking, std::string("true"));
00398     unsafeSetIfNotExist(Level::Global, ConfigurationType::MaxLogFileSize, std::string("0"));
00399     unsafeSetIfNotExist(Level::Global, ConfigurationType::Format, std::string("%datetime %level
00400 [%logger] %msg"));
00401     unsafeSetIfNotExist(Level::Debug, ConfigurationType::Format,
00402         std::string("%datetime %level [%logger] [%user@%host] [%func] [%loc] %msg"));
00403     // INFO and WARNING are set to default by Level::Global
00404     unsafeSetIfNotExist(Level::Error, ConfigurationType::Format, std::string("%datetime %level [%logger]

```

```

    %msg"));
00401     unsafeSetIfNotExist(Level::Fatal, ConfigurationType::Format, std::string("%datetime %level [%logger]
    %msg"));
00402     unsafeSetIfNotExist(Level::Verbose, ConfigurationType::Format, std::string("%datetime %level-%vlevel
    [%logger] %msg"));
00403     unsafeSetIfNotExist(Level::Trace, ConfigurationType::Format,
00404         std::string("%datetime %level [%logger] [%func] [%loc] %msg"));
00405 }
00406
00407 bool Configurations::Parser::parseFromFile(const std::string& configurationFile, Configurations*
    sender,
00408     Configurations* base) {
00409     sender->setFromBase(base);
00410     std::ifstream fileStream_(configurationFile.c_str(), std::ifstream::in);
00411     ELPP_ASSERT(fileStream_.is_open(), "Unable to open configuration file [" « configurationFile « "]
    for parsing.");
00412     bool parsedSuccessfully = false;
00413     std::string line = std::string();
00414     Level currLevel = Level::Unknown;
00415     std::string currConfigStr = std::string();
00416     std::string currLevelStr = std::string();
00417     while (fileStream_.good()) {
00418         std::getline(fileStream_, line);
00419         parsedSuccessfully = parseLine(&line, &currConfigStr, &currLevelStr, &currLevel, sender);
00420         ELPP_ASSERT(parsedSuccessfully, "Unable to parse configuration line: " « line);
00421     }
00422     return parsedSuccessfully;
00423 }
00424
00425 bool Configurations::Parser::parseFromText(const std::string& configurationsString, Configurations*
    sender,
00426     Configurations* base) {
00427     sender->setFromBase(base);
00428     bool parsedSuccessfully = false;
00429     std::stringstream ss(configurationsString);
00430     std::string line = std::string();
00431     Level currLevel = Level::Unknown;
00432     std::string currConfigStr = std::string();
00433     std::string currLevelStr = std::string();
00434     while (std::getline(ss, line)) {
00435         parsedSuccessfully = parseLine(&line, &currConfigStr, &currLevelStr, &currLevel, sender);
00436         ELPP_ASSERT(parsedSuccessfully, "Unable to parse configuration line: " « line);
00437     }
00438     return parsedSuccessfully;
00439 }
00440
00441 void Configurations::Parser::ignoreComments(std::string* line) {
00442     std::size_t foundAt = 0;
00443     std::size_t quotesStart = line->find("\"");
00444     std::size_t quotesEnd = std::string::npos;
00445     if (quotesStart != std::string::npos) {
00446         quotesEnd = line->find("\"", quotesStart + 1);
00447         while (quotesEnd != std::string::npos && line->at(quotesEnd - 1) == '\\') {
00448             // Do not erase slash yet - we will erase it in parseLine(..) while loop
00449             quotesEnd = line->find("\"", quotesEnd + 2);
00450         }
00451     }
00452     if ((foundAt = line->find(base::consts::kConfigurationComment)) != std::string::npos) {
00453         if (foundAt < quotesEnd) {
00454             foundAt = line->find(base::consts::kConfigurationComment, quotesEnd + 1);
00455         }
00456         *line = line->substr(0, foundAt);
00457     }
00458 }
00459
00460 bool Configurations::Parser::isLevel(const std::string& line) {
00461     return base::utils::Str::startsWith(line, std::string(base::consts::kConfigurationLevel));
00462 }
00463
00464 bool Configurations::Parser::isComment(const std::string& line) {
00465     return base::utils::Str::startsWith(line, std::string(base::consts::kConfigurationComment));
00466 }
00467
00468 bool Configurations::Parser::isConfig(const std::string& line) {
00469     std::size_t assignment = line.find('=');
00470     return line != "" &&
00471         ((line[0] >= 'A' && line[0] <= 'Z') || (line[0] >= 'a' && line[0] <= 'z')) &&
00472         (assignment != std::string::npos) &&
00473         (line.size() > assignment);
00474 }
00475
00476 bool Configurations::Parser::parseLine(std::string* line, std::string* currConfigStr, std::string*
    currLevelStr,
00477     Level* currLevel,
00478     Configurations* conf) {
00479     ConfigurationType currConfig = ConfigurationType::Unknown;
00480     std::string currValue = std::string();

```

```

00481     *line = base::utils::Str::trim(*line);
00482     if (isComment(*line)) return true;
00483     ignoreComments(line);
00484     *line = base::utils::Str::trim(*line);
00485     if (line->empty()) {
00486         // Comment ignored
00487         return true;
00488     }
00489     if (isLevel(*line)) {
00490         if (line->size() <= 2) {
00491             return true;
00492         }
00493         *currLevelStr = line->substr(1, line->size() - 2);
00494         *currLevelStr = base::utils::Str::toUpper(*currLevelStr);
00495         *currLevelStr = base::utils::Str::trim(*currLevelStr);
00496         *currLevel = LevelHelper::convertFromString(currLevelStr->c_str());
00497         return true;
00498     }
00499     if (isConfig(*line)) {
00500         std::size_t assignment = line->find('=');
00501         *currConfigStr = line->substr(0, assignment);
00502         *currConfigStr = base::utils::Str::toUpper(*currConfigStr);
00503         *currConfigStr = base::utils::Str::trim(*currConfigStr);
00504         *currConfig = ConfigurationTypeHelper::convertFromString(currConfigStr->c_str());
00505         currValue = line->substr(assignment + 1);
00506         currValue = base::utils::Str::trim(currValue);
00507         std::size_t quotesStart = currValue.find("\"", 0);
00508         std::size_t quotesEnd = std::string::npos;
00509         if (quotesStart != std::string::npos) {
00510             quotesEnd = currValue.find("\"", quotesStart + 1);
00511             while (quotesEnd != std::string::npos && currValue.at(quotesEnd - 1) == '\\') {
00512                 currValue = currValue.erase(quotesEnd - 1, 1);
00513                 quotesEnd = currValue.find("\"", quotesEnd + 2);
00514             }
00515         }
00516         if (quotesStart != std::string::npos && quotesEnd != std::string::npos) {
00517             // Quote provided - check and strip if valid
00518             ELPP_ASSERT((quotesStart < quotesEnd), "Configuration error - No ending quote found in ["
00519                 « currConfigStr « "]");
00520             ELPP_ASSERT((quotesStart + 1 != quotesEnd), "Empty configuration value for [" « currConfigStr «
00521 "]");
00522             if ((quotesStart != quotesEnd) && (quotesStart + 1 != quotesEnd)) {
00523                 // Explicit check in case if assertion is disabled
00524                 currValue = currValue.substr(quotesStart + 1, quotesEnd - 1);
00525             }
00526         }
00527         ELPP_ASSERT(*currLevel != Level::Unknown, "Unrecognized severity level [" « *currLevelStr « "]");
00528         ELPP_ASSERT(*currConfig != ConfigurationType::Unknown, "Unrecognized configuration [" «
00529 *currConfigStr « "]");
00530         if (*currLevel == Level::Unknown || *currConfig == ConfigurationType::Unknown) {
00531             return false; // unrecognizable level or config
00532         }
00533         *currConfig->set(*currLevel, *currConfig, currValue);
00534         return true;
00535     }
00536 void Configurations::unsafeSetIfNotExist(Level level, ConfigurationType configurationType, const
00537 std::string& value) {
00538     Configuration* conf = RegistryWithPred<Configuration, Configuration::Predicate>::get(level,
00539 configurationType);
00540     if (conf == nullptr) {
00541         unsafeSet(level, configurationType, value);
00542     }
00543 void Configurations::unsafeSet(Level level, ConfigurationType configurationType, const std::string&
00544 value) {
00545     Configuration* conf = RegistryWithPred<Configuration, Configuration::Predicate>::get(level,
00546 configurationType);
00547     if (conf == nullptr) {
00548         registerNew(new Configuration(level, configurationType, value));
00549     } else {
00550         *conf->setValue(value);
00551     }
00552     if (level == Level::Global) {
00553         unsafeSetGlobally(configurationType, value, false);
00554     }
00555 void Configurations::setGlobally(ConfigurationType configurationType, const std::string& value,
00556 bool includeGlobalLevel) {
00557     if (includeGlobalLevel) {
00558         set(Level::Global, configurationType, value);
00559     }
00560     base::type::EnumType lIndex = LevelHelper::kMinValid;
00561     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {

```

```

00562     set(LevelHelper::castFromInt(lIndex), configurationType, value);
00563     return false; // Do not break lambda function yet as we need to set all levels regardless
00564 });
00565 }
00566
00567 void Configurations::unsafeSetGlobally(ConfigurationType configurationType, const std::string& value,
00568                                     bool includeGlobalLevel) {
00569     if (includeGlobalLevel) {
00570         unsafeSet(Level::Global, configurationType, value);
00571     }
00572     base::type::EnumType lIndex = LevelHelper::kMinValid;
00573     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
00574         unsafeSet(LevelHelper::castFromInt(lIndex), configurationType, value);
00575         return false; // Do not break lambda function yet as we need to set all levels regardless
00576     });
00577 }
00578
00579 // LogBuilder
00580
00581 void LogBuilder::convertToColoredOutput(base::type::string_t* logLine, Level level) {
00582     if (!m_termSupportsColor) return;
00583     const base::type::char_t* resetColor = ELPP_LITERAL("\x1b[0m");
00584     if (level == Level::Error || level == Level::Fatal)
00585         *logLine = ELPP_LITERAL("\x1b[31m") + *logLine + resetColor;
00586     else if (level == Level::Warning)
00587         *logLine = ELPP_LITERAL("\x1b[33m") + *logLine + resetColor;
00588     else if (level == Level::Debug)
00589         *logLine = ELPP_LITERAL("\x1b[32m") + *logLine + resetColor;
00590     else if (level == Level::Info)
00591         *logLine = ELPP_LITERAL("\x1b[36m") + *logLine + resetColor;
00592     else if (level == Level::Trace)
00593         *logLine = ELPP_LITERAL("\x1b[35m") + *logLine + resetColor;
00594 }
00595
00596 // Logger
00597
00598 Logger::Logger(const std::string& id, base::LogStreamsReferenceMapPtr logStreamsReference) :
00599     m_id(id),
00600     m_typedConfigurations(nullptr),
00601     m_parentApplicationName(std::string()),
00602     m_isConfigured(false),
00603     m_logStreamsReference(logStreamsReference) {
00604     initUnflushedCount();
00605 }
00606
00607 Logger::Logger(const std::string& id, const Configurations& configurations,
00608               base::LogStreamsReferenceMapPtr logStreamsReference) :
00609     m_id(id),
00610     m_typedConfigurations(nullptr),
00611     m_parentApplicationName(std::string()),
00612     m_isConfigured(false),
00613     m_logStreamsReference(logStreamsReference) {
00614     initUnflushedCount();
00615     configure(configurations);
00616 }
00617
00618 Logger::Logger(const Logger& logger) {
00619     base::utils::safeDelete(m_typedConfigurations);
00620     m_id = logger.m_id;
00621     m_typedConfigurations = logger.m_typedConfigurations;
00622     m_parentApplicationName = logger.m_parentApplicationName;
00623     m_isConfigured = logger.m_isConfigured;
00624     m_configurations = logger.m_configurations;
00625     m_unflushedCount = logger.m_unflushedCount;
00626     m_logStreamsReference = logger.m_logStreamsReference;
00627 }
00628
00629 Logger& Logger::operator=(const Logger& logger) {
00630     if (&logger != this) {
00631         base::utils::safeDelete(m_typedConfigurations);
00632         m_id = logger.m_id;
00633         m_typedConfigurations = logger.m_typedConfigurations;
00634         m_parentApplicationName = logger.m_parentApplicationName;
00635         m_isConfigured = logger.m_isConfigured;
00636         m_configurations = logger.m_configurations;
00637         m_unflushedCount = logger.m_unflushedCount;
00638         m_logStreamsReference = logger.m_logStreamsReference;
00639     }
00640     return *this;
00641 }
00642
00643 void Logger::configure(const Configurations& configurations) {
00644     m_isConfigured = false; // we set it to false in case if we fail
00645     initUnflushedCount();
00646     if (m_typedConfigurations != nullptr) {
00647         Configurations* c = const_cast<Configurations*>(m_typedConfigurations->configurations());
00648         if (c->hasConfiguration(Level::Global, ConfigurationType::Filename)) {

```

```

00649     flush();
00650 }
00651 }
00652 base::threading::ScopedLock scopedLock(lock());
00653 if (m_configurations != configurations) {
00654     m_configurations.setFromBase(const_cast<Configurations*>(&configurations));
00655 }
00656 base::utils::safeDelete(m_typedConfigurations);
00657 m_typedConfigurations = new base::TypedConfigurations(&m_configurations, m_logStreamsReference);
00658 resolveLoggerFormatSpec();
00659 m_isConfigured = true;
00660 }
00661
00662 void Logger::reconfigure(void) {
00663     ELPP_INTERNAL_INFO(1, "Reconfiguring logger [" < m_id < "]);
00664     configure(m_configurations);
00665 }
00666
00667 bool Logger::isValidId(const std::string& id) {
00668     for (std::string::const_iterator it = id.begin(); it != id.end(); ++it) {
00669         if (!base::utils::Str::contains(base::consts::kValidLoggerIdSymbols, *it)) {
00670             return false;
00671         }
00672     }
00673     return true;
00674 }
00675
00676 void Logger::flush(void) {
00677     ELPP_INTERNAL_INFO(3, "Flushing logger [" < m_id < "] all levels");
00678     base::threading::ScopedLock scopedLock(lock());
00679     base::type::EnumType lIndex = LevelHelper::kMinValid;
00680     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
00681         flush(LevelHelper::castFromInt(lIndex), nullptr);
00682         return false;
00683     });
00684 }
00685
00686 void Logger::flush(Level level, base::type::fstream_t* fs) {
00687     if (fs == nullptr && m_typedConfigurations->toFile(level)) {
00688         fs = m_typedConfigurations->fileStream(level);
00689     }
00690     if (fs != nullptr) {
00691         fs->flush();
00692         std::unordered_map<Level, unsigned int>::iterator iter = m_unflushedCount.find(level);
00693         if (iter != m_unflushedCount.end()) {
00694             iter->second = 0;
00695         }
00696         Helpers::validateFileRolling(this, level);
00697     }
00698 }
00699
00700 void Logger::initUnflushedCount(void) {
00701     m_unflushedCount.clear();
00702     base::type::EnumType lIndex = LevelHelper::kMinValid;
00703     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
00704         m_unflushedCount.insert(std::make_pair(LevelHelper::castFromInt(lIndex), 0));
00705         return false;
00706     });
00707 }
00708
00709 void Logger::resolveLoggerFormatSpec(void) const {
00710     base::type::EnumType lIndex = LevelHelper::kMinValid;
00711     LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
00712         base::LogFormat* logFormat =
00713             const_cast<base::LogFormat*>(&m_typedConfigurations->logFormat(LevelHelper::castFromInt(lIndex)));
00714         base::utils::Str::replaceFirstWithEscape(logFormat->m_format,
00715         base::consts::kLoggerIdFormatSpecifier, m_id);
00716         return false;
00717     });
00718 }
00719 // el::base
00720 namespace base {
00721
00722 // el::base::utils
00723 namespace utils {
00724
00725 // File
00726
00727 base::type::fstream_t* File::newFileStream(const std::string& filename) {
00728     base::type::fstream_t *fs = new base::type::fstream_t(filename.c_str(),
00729     base::type::fstream_t::out
00730 #if !defined(ELPP_FRESH_LOG_FILE)
00731     | base::type::fstream_t::app
00732 #endif
00733     );
00734 #if defined(ELPP_UNICODE)

```

```

00735     std::locale elppUnicodeLocale("");
00736 #   if ELPP_OS_WINDOWS
00737     std::locale elppUnicodeLocaleWindows(elppUnicodeLocale, new std::codecvt_utf8_utf16<wchar_t>);
00738     elppUnicodeLocale = elppUnicodeLocaleWindows;
00739 #   endif // ELPP_OS_WINDOWS
00740     fs->imbue(elppUnicodeLocale);
00741 #endif // defined(ELPP_UNICODE)
00742     if (fs->is_open()) {
00743         fs->flush();
00744     } else {
00745         base::utils::safeDelete(fs);
00746         ELPP_INTERNAL_ERROR("Bad file [" « filename « "]", true);
00747     }
00748     return fs;
00749 }
00750
00751 std::size_t File::getSizeOfFile(base::type::fstream_t* fs) {
00752     if (fs == nullptr) {
00753         return 0;
00754     }
00755     // Since the file stream is appended to or truncated, the current
00756     // offset is the file size.
00757     std::size_t size = static_cast<std::size_t>(fs->tellg());
00758     return size;
00759 }
00760
00761 bool File::pathExists(const char* path, bool considerFile) {
00762     if (path == nullptr) {
00763         return false;
00764     }
00765 #if ELPP_OS_UNIX
00766     ELPP_UNUSED(considerFile);
00767     struct stat st;
00768     return (stat(path, &st) == 0);
00769 #elif ELPP_OS_WINDOWS
00770     DWORD fileType = GetFileAttributesA(path);
00771     if (fileType == INVALID_FILE_ATTRIBUTES) {
00772         return false;
00773     }
00774     return considerFile ? true : ((fileType & FILE_ATTRIBUTE_DIRECTORY) == 0 ? false : true);
00775 #endif // ELPP_OS_UNIX
00776 }
00777
00778 bool File::createPath(const std::string& path) {
00779     if (path.empty()) {
00780         return false;
00781     }
00782     if (base::utils::File::pathExists(path.c_str())) {
00783         return true;
00784     }
00785     int status = -1;
00786
00787     char* currPath = const_cast<char*>(path.c_str());
00788     std::string builtPath = std::string();
00789 #if ELPP_OS_UNIX
00790     if (path[0] == '/') {
00791         builtPath = "/";
00792     }
00793     currPath = STRTOK(currPath, base::consts::kFilePathSeparator, 0);
00794 #elif ELPP_OS_WINDOWS
00795     // Use secure functions API
00796     char* nextTok_ = nullptr;
00797     currPath = STRTOK(currPath, base::consts::kFilePathSeparator, &nextTok_);
00798     ELPP_UNUSED(nextTok_);
00799 #endif // ELPP_OS_UNIX
00800     while (currPath != nullptr) {
00801         builtPath.append(currPath);
00802         builtPath.append(base::consts::kFilePathSeparator);
00803 #if ELPP_OS_UNIX
00804         status = mkdir(builtPath.c_str(), ELPP_LOG_PERMS);
00805         currPath = STRTOK(nullptr, base::consts::kFilePathSeparator, 0);
00806 #elif ELPP_OS_WINDOWS
00807         status = _mkdir(builtPath.c_str());
00808         currPath = STRTOK(nullptr, base::consts::kFilePathSeparator, &nextTok_);
00809 #endif // ELPP_OS_UNIX
00810     }
00811     if (status == -1) {
00812         ELPP_INTERNAL_ERROR("Error while creating path [" « path « "]", true);
00813         return false;
00814     }
00815     return true;
00816 }
00817
00818 std::string File::extractPathFromFilename(const std::string& fullPath, const char* separator) {
00819     if ((fullPath == "") || (fullPath.find(separator) == std::string::npos)) {
00820         return fullPath;
00821     }

```



```

00822     std::size_t lastSlashAt = fullPath.find_last_of(separator);
00823     if (lastSlashAt == 0) {
00824         return std::string(separator);
00825     }
00826     return fullPath.substr(0, lastSlashAt + 1);
00827 }
00828
00829 void File::buildStrippedFilename(const char* filename, char buff[], std::size_t limit) {
00830     std::size_t sizeOfFilename = strlen(filename);
00831     if (sizeOfFilename >= limit) {
00832         filename += (sizeOfFilename - limit);
00833         if (filename[0] != '.' && filename[1] != '.') { // prepend if not already
00834             filename += 3; // 3 = '..'
00835             STRCAT(buff, "..", limit);
00836         }
00837     }
00838     STRCAT(buff, filename, limit);
00839 }
00840
00841 void File::buildBaseFilename(const std::string& fullPath, char buff[], std::size_t limit, const char*
separator) {
00842     const char *filename = fullPath.c_str();
00843     std::size_t lastSlashAt = fullPath.find_last_of(separator);
00844     filename += lastSlashAt ? lastSlashAt+1 : 0;
00845     std::size_t sizeOfFilename = strlen(filename);
00846     if (sizeOfFilename >= limit) {
00847         filename += (sizeOfFilename - limit);
00848         if (filename[0] != '.' && filename[1] != '.') { // prepend if not already
00849             filename += 3; // 3 = '..'
00850             STRCAT(buff, "..", limit);
00851         }
00852     }
00853     STRCAT(buff, filename, limit);
00854 }
00855
00856 // Str
00857
00858 bool Str::wildCardMatch(const char* str, const char* pattern) {
00859     while (*pattern) {
00860         switch (*pattern) {
00861             case '?':
00862                 if (!*str)
00863                     return false;
00864                 ++str;
00865                 ++pattern;
00866                 break;
00867             case '*':
00868                 if (wildCardMatch(str, pattern + 1))
00869                     return true;
00870                 if (*str && wildCardMatch(str + 1, pattern))
00871                     return true;
00872                 return false;
00873             default:
00874                 if (*str++ != *pattern++)
00875                     return false;
00876                 break;
00877         }
00878     }
00879     return !*str && !*pattern;
00880 }
00881
00882 std::string& Str::ltrim(std::string& str) {
00883     str.erase(str.begin(), std::find_if(str.begin(), str.end(), [](char c) {
00884         return !std::isspace(c);
00885     }));
00886     return str;
00887 }
00888
00889 std::string& Str:: rtrim(std::string& str) {
00890     str.erase(std::find_if(str.rbegin(), str.rend(), [](char c) {
00891         return !std::isspace(c);
00892     }).base(), str.end());
00893     return str;
00894 }
00895
00896 std::string& Str::trim(std::string& str) {
00897     return ltrim(rtrim(str));
00898 }
00899
00900 bool Str::startsWith(const std::string& str, const std::string& start) {
00901     return (str.length() >= start.length()) && (str.compare(0, start.length(), start) == 0);
00902 }
00903
00904 bool Str::endsWith(const std::string& str, const std::string& end) {
00905     return (str.length() >= end.length()) && (str.compare(str.length() - end.length(), end.length(),
end) == 0);
00906 }

```

```

00907
00908 std::string& Str::replaceAll(std::string& str, char replaceWhat, char replaceWith) {
00909     std::replace(str.begin(), str.end(), replaceWhat, replaceWith);
00910     return str;
00911 }
00912
00913 std::string& Str::replaceAll(std::string& str, const std::string& replaceWhat,
00914                             const std::string& replaceWith) {
00915     if (replaceWhat == replaceWith)
00916         return str;
00917     std::size_t foundAt = std::string::npos;
00918     while ((foundAt = str.find(replaceWhat, foundAt + 1)) != std::string::npos) {
00919         str.replace(foundAt, replaceWhat.length(), replaceWith);
00920     }
00921     return str;
00922 }
00923
00924 void Str::replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t& replaceWhat,
00925                                const base::type::string_t& replaceWith) {
00926     std::size_t foundAt = base::type::string_t::npos;
00927     while ((foundAt = str.find(replaceWhat, foundAt + 1)) != base::type::string_t::npos) {
00928         if (foundAt > 0 && str[foundAt - 1] == base::consts::kFormatSpecifierChar) {
00929             str.erase(foundAt - 1, 1);
00930             ++foundAt;
00931         } else {
00932             str.replace(foundAt, replaceWhat.length(), replaceWith);
00933             return;
00934         }
00935     }
00936 }
00937 #if defined(ELPP_UNICODE)
00938 void Str::replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t& replaceWhat,
00939                                 const std::string& replaceWith) {
00940     replaceFirstWithEscape(str, replaceWhat, base::type::string_t(replaceWith.begin(),
00941                             replaceWith.end()));
00942 }
00943 #endif // defined(ELPP_UNICODE)
00944
00945 std::string& Str::toUpper(std::string& str) {
00946     std::transform(str.begin(), str.end(), str.begin(),
00947         [](char c) {
00948             return static_cast<char> (::toupper(c));
00949         });
00950     return str;
00951 }
00952
00953 bool Str::cStringEq(const char* s1, const char* s2) {
00954     if (s1 == nullptr && s2 == nullptr) return true;
00955     if (s1 == nullptr || s2 == nullptr) return false;
00956     return strcmp(s1, s2) == 0;
00957 }
00958
00959 bool Str::cStringCaseEq(const char* s1, const char* s2) {
00960     if (s1 == nullptr && s2 == nullptr) return true;
00961     if (s1 == nullptr || s2 == nullptr) return false;
00962     // With thanks to cygwin for this code
00963     int d = 0;
00964     while (true) {
00965         const int c1 = toupper(*s1++);
00966         const int c2 = toupper(*s2++);
00967         if ((d = c1 - c2) != 0 || (c2 == '\0')) {
00968             break;
00969         }
00970     }
00971     return d == 0;
00972 }
00973
00974 bool Str::contains(const char* str, char c) {
00975     for (; *str; ++str) {
00976         if (*str == c)
00977             return true;
00978     }
00979     return false;
00980 }
00981
00982 char* Str::convertAndAddToBuff(std::size_t n, int len, char* buf, const char* bufLim, bool zeroPadded)
00983 {
00984     char localBuff[10] = "";
00985     char* p = localBuff + sizeof(localBuff) - 2;
00986     if (n > 0) {
00987         for (; n > 0 && p > localBuff && len > 0; n /= 10, --len)
00988             *--p = static_cast<char>(n % 10 + '0');
00989     } else {

```

```

00992     *--p = '0';
00993     --len;
00994 }
00995 if (zeroPadded)
00996     while (p > localBuff && len-- > 0) *--p = static_cast<char>('0');
00997 return addToBuff(p, buf, bufLim);
00998 }
00999
01000 char* Str::addToBuff(const char* str, char* buf, const char* bufLim) {
01001     while ((buf < bufLim) && ((*buf = *str++) != '\0'))
01002         ++buf;
01003     return buf;
01004 }
01005
01006 char* Str::clearBuff(char buff[], std::size_t lim) {
01007     STRCPY(buff, "", lim);
01008     ELPP_UNUSED(lim); // For *nix we dont have anything using lim in above STRCPY macro
01009     return buff;
01010 }
01011
01014 char* Str::wcharPtrToCharPtr(const wchar_t* line) {
01015     std::size_t len_ = wcslen(line) + 1;
01016     char* buff_ = static_cast<char*>(malloc(len_ + 1));
01017     # if ELPP_OS_UNIX || (ELPP_OS_WINDOWS && !ELPP_CRT_DBG_WARNINGS)
01018     std::wcstombs(buff_, line, len_);
01019     # elif ELPP_OS_WINDOWS
01020     std::size_t convCount_ = 0;
01021     mbstate_t mbState_;
01022     ::memset(static_cast<void*>(&mbState_), 0, sizeof(mbState_));
01023     wcstombs_s(&convCount_, buff_, len_, &line, len_, &mbState_);
01024     # endif // ELPP_OS_UNIX || (ELPP_OS_WINDOWS && !ELPP_CRT_DBG_WARNINGS)
01025     return buff_;
01026 }
01027
01028 // OS
01029
01030 #if ELPP_OS_WINDOWS
01035 const char* OS::getWindowsEnvironmentVariable(const char* varname) {
01036     const DWORD bufferLen = 50;
01037     static char buffer[bufferLen];
01038     if (GetEnvironmentVariableA(varname, buffer, bufferLen)) {
01039         return buffer;
01040     }
01041     return nullptr;
01042 }
01043 #endif // ELPP_OS_WINDOWS
01044 #if ELPP_OS_ANDROID
01045 std::string OS::getProperty(const char* prop) {
01046     char propVal[PROP_VALUE_MAX + 1];
01047     int ret = __system_property_get(prop, propVal);
01048     return ret == 0 ? std::string() : std::string(propVal);
01049 }
01050
01051 std::string OS::getDeviceName(void) {
01052     std::stringstream ss;
01053     std::string manufacturer = getProperty("ro.product.manufacturer");
01054     std::string model = getProperty("ro.product.model");
01055     if (manufacturer.empty() || model.empty()) {
01056         return std::string();
01057     }
01058     ss << manufacturer << "-" << model;
01059     return ss.str();
01060 }
01061 #endif // ELPP_OS_ANDROID
01062
01063 const std::string OS::getBashOutput(const char* command) {
01064     #if (ELPP_OS_UNIX && !ELPP_OS_ANDROID && !ELPP_CYGWIN)
01065     if (command == nullptr) {
01066         return std::string();
01067     }
01068     FILE* proc = nullptr;
01069     if ((proc = popen(command, "r")) == nullptr) {
01070         ELPP_INTERNAL_ERROR("\nUnable to run command [" << command << "]", true);
01071         return std::string();
01072     }
01073     char hBuff[4096];
01074     if (fgets(hBuff, sizeof(hBuff), proc) != nullptr) {
01075         pclose(proc);
01076         const std::size_t buffLen = strlen(hBuff);
01077         if (buffLen > 0 && hBuff[buffLen - 1] == '\n') {
01078             hBuff[buffLen - 1] = '\0';
01079         }
01080         return std::string(hBuff);
01081     } else {
01082         pclose(proc);
01083     }
01084     return std::string();

```

```

01085 #else
01086     ELPP_UNUSED(command);
01087     return std::string();
01088 #endif // (ELPP_OS_UNIX && !ELPP_OS_ANDROID && !ELPP_CYGWIN)
01089 }
01090
01091 std::string OS::getEnvironmentVariable(const char* variableName, const char* defaultVal,
01092                                       const char* alternativeBashCommand) {
01093     #if ELPP_OS_UNIX
01094         const char* val = getenv(variableName);
01095     #elif ELPP_OS_WINDOWS
01096         const char* val = getWindowsEnvironmentVariable(variableName);
01097     #endif // ELPP_OS_UNIX
01098     if ((val == nullptr) || ((strcmp(val, "") == 0))) {
01099     #if ELPP_OS_UNIX && defined(ELPP_FORCE_ENV_VAR_FROM_BASH)
01100         // Try harder on unix-based systems
01101         std::string valBash = base::utils::OS::getBashOutput(alternativeBashCommand);
01102         if (valBash.empty()) {
01103             return std::string(defaultVal);
01104         } else {
01105             return valBash;
01106         }
01107     #elif ELPP_OS_WINDOWS || ELPP_OS_UNIX
01108         ELPP_UNUSED(alternativeBashCommand);
01109         return std::string(defaultVal);
01110     #endif // ELPP_OS_UNIX && defined(ELPP_FORCE_ENV_VAR_FROM_BASH)
01111     }
01112     return std::string(val);
01113 }
01114
01115 std::string OS::currentUser(void) {
01116     #if ELPP_OS_UNIX && !ELPP_OS_ANDROID
01117         return getEnvironmentVariable("USER", base::consts::kUnknownUser, "whoami");
01118     #elif ELPP_OS_WINDOWS
01119         return getEnvironmentVariable("USERNAME", base::consts::kUnknownUser);
01120     #elif ELPP_OS_ANDROID
01121         ELPP_UNUSED(base::consts::kUnknownUser);
01122         return std::string("android");
01123     #else
01124         return std::string();
01125     #endif // ELPP_OS_UNIX && !ELPP_OS_ANDROID
01126 }
01127
01128 std::string OS::currentHost(void) {
01129     #if ELPP_OS_UNIX && !ELPP_OS_ANDROID
01130         return getEnvironmentVariable("HOSTNAME", base::consts::kUnknownHost, "hostname");
01131     #elif ELPP_OS_WINDOWS
01132         return getEnvironmentVariable("COMPUTERNAME", base::consts::kUnknownHost);
01133     #elif ELPP_OS_ANDROID
01134         ELPP_UNUSED(base::consts::kUnknownHost);
01135         return getDeviceName();
01136     #else
01137         return std::string();
01138     #endif // ELPP_OS_UNIX && !ELPP_OS_ANDROID
01139 }
01140
01141 bool OS::termSupportsColor(void) {
01142     std::string term = getEnvironmentVariable("TERM", "");
01143     return term == "xterm" || term == "xterm-color" || term == "xterm-256color"
01144         || term == "screen" || term == "linux" || term == "cygwin"
01145         || term == "screen-256color";
01146 }
01147
01148 // DateTime
01149
01150 void DateTime::gettimeofday(struct timeval* tv) {
01151     #if ELPP_OS_WINDOWS
01152         if (tv != nullptr) {
01153             # if ELPP_COMPILER_MSVC || defined(_MSC_EXTENSIONS)
01154                 const unsigned __int64 delta_ = 1164447360000000000ui64;
01155             # else
01156                 const unsigned __int64 delta_ = 1164447360000000000ULL;
01157             # endif // ELPP_COMPILER_MSVC || defined(_MSC_EXTENSIONS)
01158             const double secOffSet = 0.000001;
01159             const unsigned long usecOffSet = 1000000;
01160             FILETIME fileTime;
01161             GetSystemTimeAsFileTime(&fileTime);
01162             unsigned __int64 present = 0;
01163             present |= fileTime.dwHighDateTime;
01164             present = present << 32;
01165             present |= fileTime.dwLowDateTime;
01166             present /= 10; // mic-sec
01167             // Subtract the difference
01168             present -= delta_;
01169             tv->tv_sec = static_cast<long>(present * secOffSet);
01170             tv->tv_usec = static_cast<long>(present % usecOffSet);
01171         }

```

```

01172 #else
01173     :gettimeofday(tv, nullptr);
01174 #endif // ELPP_OS_WINDOWS
01175 }
01176
01177 std::string DateTime::getDateTime(const char* format, const base::SubsecondPrecision* ssPrec) {
01178     struct timeval currTime;
01179     gettimeofday(&currTime);
01180     return timevalToString(currTime, format, ssPrec);
01181 }
01182
01183 std::string DateTime::timevalToString(struct timeval tval, const char* format,
01184                                     const el::base::SubsecondPrecision* ssPrec) {
01185     struct ::tm timeInfo;
01186     buildTimeInfo(&tval, &timeInfo);
01187     const int kBuffSize = 30;
01188     char buff_[kBuffSize] = "";
01189     parseFormat(buff_, kBuffSize, format, &timeInfo, static_cast<std::size_t>(tval.tv_usec /
01190 ssPrec->m_offset),
01191 ssPrec);
01192     return std::string(buff_);
01193 }
01194
01195 base::type::string_t DateTime::formatTime(unsigned long long time, base::TimestampUnit timestampUnit)
01196 {
01197     base::type::EnumType start = static_cast<base::type::EnumType>(timestampUnit);
01198     const base::type::char_t* unit = base::consts::kTimeFormats[start].unit;
01199     for (base::type::EnumType i = start; i < base::consts::kTimeFormatsCount - 1; ++i) {
01200         if (time <= base::consts::kTimeFormats[i].value) {
01201             break;
01202         }
01203         if (base::consts::kTimeFormats[i].value == 1000.0f && time / 1000.0f < 1.9f) {
01204             break;
01205         }
01206         time /= static_cast<decltype(time)>(base::consts::kTimeFormats[i].value);
01207         unit = base::consts::kTimeFormats[i + 1].unit;
01208     }
01209     base::type::stringstream_t ss;
01210     ss << time << " " << unit;
01211     return ss.str();
01212 }
01213
01214 unsigned long long DateTime::getTimeDifference(const struct timeval& endTime, const struct timeval&
01215 startTime,
01216 base::TimestampUnit timestampUnit) {
01217     if (timestampUnit == base::TimestampUnit::Microsecond) {
01218         return static_cast<unsigned long long>(static_cast<unsigned long long>(1000000 * endTime.tv_sec +
01219 endTime.tv_usec) -
01220 static_cast<unsigned long long>(1000000 * startTime.tv_sec
01221 + startTime.tv_usec));
01222     }
01223     // milliseconds
01224     auto conv = [](const struct timeval& tim) {
01225         return static_cast<unsigned long long>((tim.tv_sec * 1000) + (tim.tv_usec / 1000));
01226     };
01227     return static_cast<unsigned long long>(conv(endTime) - conv(startTime));
01228 }
01229
01230 struct ::tm* DateTime::buildTimeInfo(struct timeval* currTime, struct ::tm* timeInfo) {
01231     #if ELPP_OS_UNIX
01232         time_t rawTime = currTime->tv_sec;
01233         ::elptime_r(&rawTime, timeInfo);
01234         return timeInfo;
01235     #else
01236         # if ELPP_COMPILER_MSVC
01237             ELPP_UNUSED(currTime);
01238             time_t t;
01239             if defined(_USE_32BIT_TIME_T)
01240                 _time32(&t);
01241             # else
01242                 _time64(&t);
01243             # endif
01244             elptime_s(timeInfo, &t);
01245         # else
01246             // For any other compilers that don't have CRT warnings issue e.g, MinGW or TDM GCC- we use
01247             different method
01248             time_t rawTime = currTime->tv_sec;
01249             struct tm* tmInf = elptime(&rawTime);
01250             *timeInfo = *tmInf;
01251             return timeInfo;
01252         # endif // ELPP_COMPILER_MSVC
01253     #endif // ELPP_OS_UNIX
01254 }
01255
01256 char* DateTime::parseFormat(char* buf, std::size_t bufSz, const char* format, const struct tm* tInfo,
01257 std::size_t msec, const base::SubsecondPrecision* ssPrec) {

```

```

01253     const char* bufLim = buf + bufSz;
01254     for (; *format; ++format) {
01255         if (*format == base::consts::kFormatSpecifierChar) {
01256             switch (++format) {
01257                 case base::consts::kFormatSpecifierChar: // Escape
01258                     break;
01259                 case '\\0': // End
01260                     --format;
01261                     break;
01262                 case 'd': // Day
01263                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_mday, 2, buf, bufLim);
01264                     continue;
01265                 case 'a': // Day of week (short)
01266                     buf = base::utils::Str::addToBuff(base::consts::kDaysAbbrev[tInfo->tm_wday], buf, bufLim);
01267                     continue;
01268                 case 'A': // Day of week (long)
01269                     buf = base::utils::Str::addToBuff(base::consts::kDays[tInfo->tm_wday], buf, bufLim);
01270                     continue;
01271                 case 'M': // month
01272                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_mon + 1, 2, buf, bufLim);
01273                     continue;
01274                 case 'b': // month (short)
01275                     buf = base::utils::Str::addToBuff(base::consts::kMonthsAbbrev[tInfo->tm_mon], buf, bufLim);
01276                     continue;
01277                 case 'B': // month (long)
01278                     buf = base::utils::Str::addToBuff(base::consts::kMonths[tInfo->tm_mon], buf, bufLim);
01279                     continue;
01280                 case 'y': // year (two digits)
01281                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_year + base::consts::kYearBase, 2, buf,
bufLim);
01282                     continue;
01283                 case 'Y': // year (four digits)
01284                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_year + base::consts::kYearBase, 4, buf,
bufLim);
01285                     continue;
01286                 case 'h': // hour (12-hour)
01287                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_hour % 12, 2, buf, bufLim);
01288                     continue;
01289                 case 'H': // hour (24-hour)
01290                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_hour, 2, buf, bufLim);
01291                     continue;
01292                 case 'm': // minute
01293                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_min, 2, buf, bufLim);
01294                     continue;
01295                 case 's': // second
01296                     buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_sec, 2, buf, bufLim);
01297                     continue;
01298                 case 'z': // subsecond part
01299                 case 'g':
01300                     buf = base::utils::Str::convertAndAddToBuff(msec, ssPrec->m_width, buf, bufLim);
01301                     continue;
01302                 case 'F': // AM/PM
01303                     buf = base::utils::Str::addToBuff((tInfo->tm_hour >= 12) ? base::consts::kPm :
base::consts::kAm, buf, bufLim);
01304                     continue;
01305                 default:
01306                     continue;
01307             }
01308         }
01309         if (buf == bufLim) break;
01310         *buf++ = *format;
01311     }
01312     return buf;
01313 }
01314
01315 // CommandLineArgs
01316
01317 void CommandLineArgs::setArgs(int argc, char** argv) {
01318     m_params.clear();
01319     m_paramsWithValue.clear();
01320     if (argc == 0 || argv == nullptr) {
01321         return;
01322     }
01323     m_argc = argc;
01324     m_argv = argv;
01325     for (int i = 1; i < m_argc; ++i) {
01326         const char* v = (strstr(m_argv[i], "="));
01327         if (v != nullptr && strlen(v) > 0) {
01328             std::string key = std::string(m_argv[i]);
01329             key = key.substr(0, key.find_first_of('='));
01330             if (hasParamWithValue(key.c_str())) {
01331                 ELPP_INTERNAL_INFO(1, "Skipping [" < key < "] arg since it already has value ["
01332                     < getParamValue(key.c_str()) < "]");
01333             } else {
01334                 m_paramsWithValue.insert(std::make_pair(key, std::string(v + 1)));
01335             }
01336         }
01337     }

```

```

01337     if (v == nullptr) {
01338         if (hasParam(m_argv[i])) {
01339             ELPP_INTERNAL_INFO(1, "Skipping [" << m_argv[i] << "] arg since it already exists");
01340         } else {
01341             m_params.push_back(std::string(m_argv[i]));
01342         }
01343     }
01344 }
01345 }
01346
01347 bool CommandLineArgs::hasParamWithValue(const char* paramKey) const {
01348     return m_paramsWithValue.find(std::string(paramKey)) != m_paramsWithValue.end();
01349 }
01350
01351 const char* CommandLineArgs::getParamValue(const char* paramKey) const {
01352     std::unordered_map<std::string, std::string>::const_iterator iter =
01353         m_paramsWithValue.find(std::string(paramKey));
01354     return iter != m_paramsWithValue.end() ? iter->second.c_str() : "";
01355 }
01356 bool CommandLineArgs::hasParam(const char* paramKey) const {
01357     return std::find(m_params.begin(), m_params.end(), std::string(paramKey)) != m_params.end();
01358 }
01359
01360 bool CommandLineArgs::empty(void) const {
01361     return m_params.empty() && m_paramsWithValue.empty();
01362 }
01363
01364 std::size_t CommandLineArgs::size(void) const {
01365     return m_params.size() + m_paramsWithValue.size();
01366 }
01367
01368 base::type::ostream_t& operator<<(base::type::ostream_t& os, const CommandLineArgs& c) {
01369     for (int i = 1; i < c.m_argc; ++i) {
01370         os << ELPP_LITERAL("[") << c.m_argv[i] << ELPP_LITERAL("]");
01371         if (i < c.m_argc - 1) {
01372             os << ELPP_LITERAL(" ");
01373         }
01374     }
01375     return os;
01376 }
01377
01378 } // namespace utils
01379
01380 // el::base::threading
01381 namespace threading {
01382 #if ELPP_THREADING_ENABLED
01383 # if ELPP_USE_STD_THREADING
01384 #     if ELPP_ASYNC_LOGGING
01385 static void msleep(int ms) {
01386     // Only when async logging enabled - this is because async is strict on compiler
01387     # if defined(ELPP_NO_SLEEP_FOR)
01388         usleep(ms * 1000);
01389     # else
01390         std::this_thread::sleep_for(std::chrono::milliseconds(ms));
01391     # endif // defined(ELPP_NO_SLEEP_FOR)
01392 # endif // ELPP_ASYNC_LOGGING
01393 # endif // ELPP_USE_STD_THREADING
01394 #endif // ELPP_THREADING_ENABLED
01395
01396 } // namespace threading
01397
01398 // el::base
01399
01400 // SubsecondPrecision
01401
01402 void SubsecondPrecision::init(int width) {
01403     if (width < 1 || width > 6) {
01404         width = base::consts::kDefaultSubsecondPrecision;
01405     }
01406     m_width = width;
01407     switch (m_width) {
01408     case 3:
01409         m_offset = 1000;
01410         break;
01411     case 4:
01412         m_offset = 100;
01413         break;
01414     case 5:
01415         m_offset = 10;
01416         break;
01417     case 6:
01418         m_offset = 1;
01419         break;
01420     default:

```

```

01423     m_offset = 1000;
01424     break;
01425 }
01426 }
01427
01428 // LogFormat
01429
01430 LogFormat::LogFormat(void) :
01431     m_level(Level::Unknown),
01432     m_userFormat(base::type::string_t()),
01433     m_format(base::type::string_t()),
01434     m_dateTimeFormat(std::string()),
01435     m_flags(0x0),
01436     m_currentUser(base::utils::OS::currentUser()),
01437     m_currentHost(base::utils::OS::currentHost()) {
01438 }
01439
01440 LogFormat::LogFormat(Level level, const base::type::string_t& format)
01441 : m_level(level), m_userFormat(format), m_currentUser(base::utils::OS::currentUser()),
01442     m_currentHost(base::utils::OS::currentHost()) {
01443     parseFromFormat(m_userFormat);
01444 }
01445
01446 LogFormat::LogFormat(const LogFormat& logFormat):
01447     m_level(logFormat.m_level),
01448     m_userFormat(logFormat.m_userFormat),
01449     m_format(logFormat.m_format),
01450     m_dateTimeFormat(logFormat.m_dateTimeFormat),
01451     m_flags(logFormat.m_flags),
01452     m_currentUser(logFormat.m_currentUser),
01453     m_currentHost(logFormat.m_currentHost) {
01454 }
01455
01456 LogFormat::LogFormat(LogFormat&& logFormat) {
01457     m_level = std::move(logFormat.m_level);
01458     m_userFormat = std::move(logFormat.m_userFormat);
01459     m_format = std::move(logFormat.m_format);
01460     m_dateTimeFormat = std::move(logFormat.m_dateTimeFormat);
01461     m_flags = std::move(logFormat.m_flags);
01462     m_currentUser = std::move(logFormat.m_currentUser);
01463     m_currentHost = std::move(logFormat.m_currentHost);
01464 }
01465
01466 LogFormat& LogFormat::operator=(const LogFormat& logFormat) {
01467     if (&logFormat != this) {
01468         m_level = logFormat.m_level;
01469         m_userFormat = logFormat.m_userFormat;
01470         m_dateTimeFormat = logFormat.m_dateTimeFormat;
01471         m_flags = logFormat.m_flags;
01472         m_currentUser = logFormat.m_currentUser;
01473         m_currentHost = logFormat.m_currentHost;
01474     }
01475     return *this;
01476 }
01477
01478 bool LogFormat::operator==(const LogFormat& other) {
01479     return m_level == other.m_level && m_userFormat == other.m_userFormat && m_format == other.m_format
01480         &&
01481         m_dateTimeFormat == other.m_dateTimeFormat && m_flags == other.m_flags;
01482 }
01483
01484 void LogFormat::parseFromFormat(const base::type::string_t& userFormat) {
01485     // We make copy because we will be changing the format
01486     // i.e, removing user provided date format from original format
01487     // and then storing it.
01488     base::type::string_t formatCopy = userFormat;
01489     m_flags = 0x0;
01490     auto conditionalAddFlag = [&](const base::type::char_t* specifier, base::FormatFlags flag) {
01491         std::size_t foundAt = base::type::string_t::npos;
01492         while ((foundAt = formatCopy.find(specifier, foundAt + 1)) != base::type::string_t::npos) {
01493             if (foundAt > 0 && formatCopy[foundAt - 1] == base::consts::kFormatSpecifierChar) {
01494                 if (hasFlag(flag)) {
01495                     // If we already have flag we remove the escape chars so that '%%' is turned to '%'
01496                     // even after specifier resolution - this is because we only replaceFirst specifier
01497                     formatCopy.erase(foundAt - 1, 1);
01498                     ++foundAt;
01499                 }
01500             } else {
01501                 if (!hasFlag(flag)) addFlag(flag);
01502             }
01503         }
01504     };
01505     conditionalAddFlag(base::consts::kAppNameFormatSpecifier, base::FormatFlags::AppName);
01506     conditionalAddFlag(base::consts::kSeverityLevelFormatSpecifier, base::FormatFlags::Level);
01507     conditionalAddFlag(base::consts::kSeverityLevelShortFormatSpecifier, base::FormatFlags::LevelShort);
01508     conditionalAddFlag(base::consts::kLoggerIdFormatSpecifier, base::FormatFlags::LoggerId);
01509     conditionalAddFlag(base::consts::kThreadIdFormatSpecifier, base::FormatFlags::ThreadId);

```



```

01511 conditionalAddFlag(base::consts::kLogFileFormatSpecifier, base::FormatFlags::File);
01512 conditionalAddFlag(base::consts::kLogFileBaseFormatSpecifier, base::FormatFlags::FileBase);
01513 conditionalAddFlag(base::consts::kLogLineFormatSpecifier, base::FormatFlags::Line);
01514 conditionalAddFlag(base::consts::kLogLocationFormatSpecifier, base::FormatFlags::Location);
01515 conditionalAddFlag(base::consts::kLogFunctionFormatSpecifier, base::FormatFlags::Function);
01516 conditionalAddFlag(base::consts::kCurrentUserFormatSpecifier, base::FormatFlags::User);
01517 conditionalAddFlag(base::consts::kCurrentHostFormatSpecifier, base::FormatFlags::Host);
01518 conditionalAddFlag(base::consts::kMessageFormatSpecifier, base::FormatFlags::LogMessage);
01519 conditionalAddFlag(base::consts::kVerboseLevelFormatSpecifier, base::FormatFlags::VerboseLevel);
01520 // For date/time we need to extract user's date format first
01521 std::size_t dateIndex = std::string::npos;
01522 if ((dateIndex = formatCopy.find(base::consts::kDateTimeFormatSpecifier)) != std::string::npos) {
01523     while (dateIndex != std::string::npos && dateIndex > 0 && formatCopy[dateIndex - 1] ==
base::consts::kFormatSpecifierChar) {
01524         dateIndex = formatCopy.find(base::consts::kDateTimeFormatSpecifier, dateIndex + 1);
01525     }
01526     if (dateIndex != std::string::npos) {
01527         addFlag(base::FormatFlags::DateTime);
01528         updateDateFormat(dateIndex, formatCopy);
01529     }
01530 }
01531 m_format = formatCopy;
01532 updateFormatSpec();
01533 }
01534
01535 void LogFormat::updateDateFormat(std::size_t index, base::type::string_t& currFormat) {
01536     if (hasFlag(base::FormatFlags::DateTime)) {
01537         index += ELPP_STRLEN(base::consts::kDateTimeFormatSpecifier);
01538     }
01539     const base::type::char_t* ptr = currFormat.c_str() + index;
01540     if ((currFormat.size() > index) && (ptr[0] == '(')) {
01541         // User has provided format for date/time
01542         ++ptr;
01543         int count = 1; // Start by 1 in order to remove starting brace
01544         std::stringstream ss;
01545         for (; *ptr; ++ptr, ++count) {
01546             if (*ptr == ')') {
01547                 ++count; // In order to remove ending brace
01548                 break;
01549             }
01550             ss << static_cast<char>(*ptr);
01551         }
01552         currFormat.erase(index, count);
01553         m_dateTimeFormat = ss.str();
01554     } else {
01555         // No format provided, use default
01556         if (hasFlag(base::FormatFlags::DateTime)) {
01557             m_dateTimeFormat = std::string(base::consts::kDefaultDateTimeFormat);
01558         }
01559     }
01560 }
01561
01562 void LogFormat::updateFormatSpec(void) {
01563     // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
01564     if (m_level == Level::Debug) {
01565         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kDebugLevelLogValue);
01566         base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
base::consts::kDebugLevelShortLogValue);
01567     } else if (m_level == Level::Info) {
01568         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kInfoLevelLogValue);
01569         base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
base::consts::kInfoLevelShortLogValue);
01570     } else if (m_level == Level::Warning) {
01571         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kWarningLevelLogValue);
01572         base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
base::consts::kWarningLevelShortLogValue);
01573     } else if (m_level == Level::Error) {
01574         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kErrorLevelLogValue);
01575         base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
base::consts::kErrorLevelShortLogValue);
01576     } else if (m_level == Level::Fatal) {
01577         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kFatalLevelLogValue);
01578         base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
base::consts::kFatalLevelShortLogValue);
01579     } else if (m_level == Level::Verbose) {
01580         base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
base::consts::kVerboseLevelLogValue);
01581     }
01582 }

```

```

01592     base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
01593         base::consts::kVerboseLevelShortLogValue);
01594 } else if (m_level == Level::Trace) {
01595     base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
01596         base::consts::kTraceLevelLogValue);
01597     base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
01598         base::consts::kTraceLevelShortLogValue);
01599 }
01600 if (hasFlag(base::FormatFlags::User)) {
01601     base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kCurrentUserFormatSpecifier,
01602         m_currentUser);
01603 }
01604 if (hasFlag(base::FormatFlags::Host)) {
01605     base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kCurrentHostFormatSpecifier,
01606         m_currentHost);
01607 }
01608 // Ignore Level::Global and Level::Unknown
01609 }
01610
01611 // TypedConfigurations
01612
01613 TypedConfigurations::TypedConfigurations(Configurations* configurations,
01614     LogStreamsReferenceMapPtr logStreamsReference) {
01615     m_configurations = configurations;
01616     m_logStreamsReference = logStreamsReference;
01617     build(m_configurations);
01618 }
01619
01620 TypedConfigurations::TypedConfigurations(const TypedConfigurations& other) {
01621     this->m_configurations = other.m_configurations;
01622     this->m_logStreamsReference = other.m_logStreamsReference;
01623     build(m_configurations);
01624 }
01625
01626 bool TypedConfigurations::enabled(Level level) {
01627     return getConfigByVal<bool>(level, &m_enabledMap, "enabled");
01628 }
01629
01630 bool TypedConfigurations::toFile(Level level) {
01631     return getConfigByVal<bool>(level, &m_toFileMap, "toFile");
01632 }
01633
01634 const std::string& TypedConfigurations::filename(Level level) {
01635     return getConfigByRef<std::string>(level, &m_filenameMap, "filename");
01636 }
01637
01638 bool TypedConfigurations::toStandardOutput(Level level) {
01639     return getConfigByVal<bool>(level, &m_toStandardOutputMap, "toStandardOutput");
01640 }
01641
01642 const base::LogFormat& TypedConfigurations::logFormat(Level level) {
01643     return getConfigByRef<base::LogFormat>(level, &m_logFormatMap, "logFormat");
01644 }
01645
01646 const base::SubsecondPrecision& TypedConfigurations::subsecondPrecision(Level level) {
01647     return getConfigByRef<base::SubsecondPrecision>(level, &m_subsecondPrecisionMap,
"subsecondPrecision");
01648 }
01649
01650 const base::MillisecondsWidth& TypedConfigurations::millisecondsWidth(Level level) {
01651     return getConfigByRef<base::MillisecondsWidth>(level, &m_subsecondPrecisionMap,
"millisecondsWidth");
01652 }
01653
01654 bool TypedConfigurations::performanceTracking(Level level) {
01655     return getConfigByVal<bool>(level, &m_performanceTrackingMap, "performanceTracking");
01656 }
01657
01658 base::type::fstream_t* TypedConfigurations::fileStream(Level level) {
01659     return getConfigByRef<base::FileStreamPtr>(level, &m_fileStreamMap, "fileStream").get();
01660 }
01661
01662 std::size_t TypedConfigurations::maxLogFileSize(Level level) {
01663     return getConfigByVal<std::size_t>(level, &m_maxLogFileSizeMap, "maxLogFileSize");
01664 }
01665
01666 std::size_t TypedConfigurations::logFlushThreshold(Level level) {
01667     return getConfigByVal<std::size_t>(level, &m_logFlushThresholdMap, "logFlushThreshold");
01668 }
01669
01670 void TypedConfigurations::build(Configurations* configurations) {
01671     base::threading::ScopedLock scopedLock(lock());
01672     auto getBool = [] (std::string boolStr) -> bool { // Pass by value for trimming
01673         base::utils::Str::trim(boolStr);
01674         return (boolStr == "TRUE" || boolStr == "true" || boolStr == "1");

```

```

01675     };
01676     std::vector<Configuration*> withFileSizeLimit;
01677     for (Configurations::const_iterator it = configurations->begin(); it != configurations->end(); ++it)
01678     {
01679         Configuration* conf = *it;
01680         // We cannot use switch on strong enums because Intel C++ dont support them yet
01681         if (conf->configurationType() == ConfigurationType::Enabled) {
01682             setValue(conf->level(), getBool(conf->value()), &m_enabledMap);
01683         } else if (conf->configurationType() == ConfigurationType::ToFile) {
01684             setValue(conf->level(), getBool(conf->value()), &m_toFileMap);
01685         } else if (conf->configurationType() == ConfigurationType::ToStandardOutput) {
01686             setValue(conf->level(), getBool(conf->value()), &m_toStandardOutputMap);
01687         } else if (conf->configurationType() == ConfigurationType::Filename) {
01688             // We do not yet configure filename but we will configure in another
01689             // loop. This is because if file cannot be created, we will force ToFile
01690             // to be false. Because configuring logger is not necessarily performance
01691             // sensitive operation, we can live with another loop; (by the way this loop
01692             // is not very heavy either)
01693         } else if (conf->configurationType() == ConfigurationType::Format) {
01694             setValue(conf->level(), base::LogFormat(conf->level(),
01695                 base::type::string_t(conf->value().begin(),
01696                     conf->value().end())), &m_logFormatMap);
01697         } else if (conf->configurationType() == ConfigurationType::SubsecondPrecision) {
01698             setValue(Level::Global,
01699                 base::SubsecondPrecision(static_cast<int>(getULong(conf->value()))),
01700                 &m_subsecondPrecisionMap);
01701         } else if (conf->configurationType() == ConfigurationType::PerformanceTracking) {
01702             setValue(Level::Global, getBool(conf->value()), &m_performanceTrackingMap);
01703         } else if (conf->configurationType() == ConfigurationType::MaxLogFileSize) {
01704             auto v = getULong(conf->value());
01705             setValue(conf->level(), static_cast<std::size_t>(v), &m_maxLogFileSizeMap);
01706             if (v != 0) {
01707                 withFileSizeLimit.push_back(conf);
01708             }
01709         } else if (conf->configurationType() == ConfigurationType::LogFlushThreshold) {
01710             setValue(conf->level(), static_cast<std::size_t>(getULong(conf->value())),
01711                 &m_logFlushThresholdMap);
01712         }
01713     }
01714     // As mentioned earlier, we will now set filename configuration in separate loop to deal with
01715     non-existent files
01716     for (Configurations::const_iterator it = configurations->begin(); it != configurations->end(); ++it)
01717     {
01718         Configuration* conf = *it;
01719         if (conf->configurationType() == ConfigurationType::Filename) {
01720             insertFile(conf->level(), conf->value());
01721         }
01722     }
01723     for (std::vector<Configuration*>::iterator conf = withFileSizeLimit.begin();
01724         conf != withFileSizeLimit.end(); ++conf) {
01725         // This is not unsafe as mutex is locked in correct scope
01726         unsafeValidateFileRolling((*conf)->level(), base::defaultPreRollOutCallback);
01727     }
01728 }
01729
01730 unsigned long TypedConfigurations::getULong(std::string confVal) {
01731     bool valid = true;
01732     base::utils::Str::trim(confVal);
01733     valid = !confVal.empty() && std::find_if(confVal.begin(), confVal.end(),
01734         [](char c) {
01735             return !base::utils::Str::isDigit(c);
01736         }) == confVal.end();
01737     if (!valid) {
01738         valid = false;
01739         ELPP_ASSERT(valid, "Configuration value not a valid integer [" << confVal << "]");
01740         return 0;
01741     }
01742     return atol(confVal.c_str());
01743 }
01744
01745 std::string TypedConfigurations::resolveFilename(const std::string& filename) {
01746     std::string resultingFilename = filename;
01747     std::size_t dateIndex = std::string::npos;
01748     std::string dateTimeFormatSpecifierStr =
01749         std::string(base::consts::kDateTimeFormatSpecifierForFilename);
01750     if ((dateIndex = resultingFilename.find(dateTimeFormatSpecifierStr.c_str())) != std::string::npos) {
01751         while (dateIndex > 0 && resultingFilename[dateIndex - 1] == base::consts::kFormatSpecifierChar) {
01752             dateIndex = resultingFilename.find(dateTimeFormatSpecifierStr.c_str(), dateIndex + 1);
01753         }
01754         if (dateIndex != std::string::npos) {
01755             const char* ptr = resultingFilename.c_str() + dateIndex;
01756             // Goto end of specifier
01757             ptr += dateTimeFormatSpecifierStr.size();
01758             std::string fmt;
01759             if ((resultingFilename.size() > dateIndex) && (ptr[0] == '{')) {
01760                 // User has provided format for date/time
01761                 ++ptr;

```

```

01755         int count = 1; // Start by 1 in order to remove starting brace
01756         std::stringstream ss;
01757         for (; *ptr; ++ptr, ++count) {
01758             if (*ptr == '}') {
01759                 ++count; // In order to remove ending brace
01760                 break;
01761             }
01762             ss << *ptr;
01763         }
01764         resultingFilename.erase(dateIndex + dateTimeFormatSpecifierStr.size(), count);
01765         fmt = ss.str();
01766     } else {
01767         fmt = std::string(base::consts::kDefaultDateTimeFormatInFilename);
01768     }
01769     base::SubsecondPrecision ssPrec(3);
01770     std::string now = base::utils::DateTime::getDateTime(fmt.c_str(), &ssPrec);
01771     base::utils::Str::replaceAll(now, '/', '-'); // Replace path element since we are dealing with
    filename
01772     base::utils::Str::replaceAll(resultingFilename, dateTimeFormatSpecifierStr, now);
01773 }
01774 }
01775 return resultingFilename;
01776 }
01777
01778 void TypedConfigurations::insertFile(Level level, const std::string& fullFilename) {
01779     std::string resolvedFilename = resolveFilename(fullFilename);
01780     if (resolvedFilename.empty()) {
01781         std::cerr << "Could not load empty file for logging, please re-check your configurations for level
    ["
01782             << LevelHelper::convertToString(level) << "];
01783     }
01784     std::string filePath = base::utils::File::extractPathFromFilename(resolvedFilename,
    base::consts::kFilePathSeparator);
01785     if (filePath.size() < resolvedFilename.size()) {
01786         base::utils::File::createPath(filePath);
01787     }
01788     auto create = [&](Level level) {
01789         base::LogStreamsReferenceMap::iterator filestreamIter =
    m_logStreamsReference->find(resolvedFilename);
01790         base::type::fstream_t* fs = nullptr;
01791         if (filestreamIter == m_logStreamsReference->end()) {
01792             // We need a completely new stream, nothing to share with
01793             fs = base::utils::File::newFileStream(resolvedFilename);
01794             m_filenameMap.insert(std::make_pair(level, resolvedFilename));
01795             m_fileStreamMap.insert(std::make_pair(level, base::FileStreamPtr(fs)));
01796             m_logStreamsReference->insert(std::make_pair(resolvedFilename,
    base::FileStreamPtr(m_fileStreamMap.at(level))));
01797         } else {
01798             // Woops! we have an existing one, share it!
01799             m_filenameMap.insert(std::make_pair(level, filestreamIter->first));
01800             m_fileStreamMap.insert(std::make_pair(level, base::FileStreamPtr(filestreamIter->second)));
01801             fs = filestreamIter->second.get();
01802         }
01803         if (fs == nullptr) {
01804             // We display bad file error from newFileStream()
01805             ELPP_INTERNAL_ERROR("Setting [TO_FILE] of ["
    << LevelHelper::convertToString(level) << "] to FALSE", false);
01806             setValue(level, false, &m_toFileMap);
01807         }
01808     };
01809 };
01810 // If we dont have file conf for any level, create it for Level::Global first
01811 // otherwise create for specified level
01812 create(m_filenameMap.empty() && m_fileStreamMap.empty() ? Level::Global : level);
01813 }
01814
01815 bool TypedConfigurations::unsafeValidateFileRolling(Level level, const PreRollOutCallback&
    preRollOutCallback) {
01816     base::type::fstream_t* fs = unsafeGetConfigByRef(level, &m_fileStreamMap, "fileStream").get();
01817     if (fs == nullptr) {
01818         return true;
01819     }
01820     std::size_t maxLogFileSize = unsafeGetConfigByVal(level, &m_maxLogFileSizeMap, "maxLogFileSize");
01821     std::size_t currFileSize = base::utils::File::getSizeOfFile(fs);
01822     if (maxLogFileSize != 0 && currFileSize >= maxLogFileSize) {
01823         std::string fname = unsafeGetConfigByRef(level, &m_filenameMap, "filename");
01824         ELPP_INTERNAL_INFO(1, "Truncating log file [" << fname << "] as a result of configurations for level
    ["
01825             << LevelHelper::convertToString(level) << "]);
01826         fs->close();
01827         preRollOutCallback(fname.c_str(), currFileSize);
01828         fs->open(fname, std::fstream::out | std::fstream::trunc);
01829         return true;
01830     }
01831     return false;
01832 }
01833
01834 // RegisteredHitCounters

```

```

01835
01836 bool RegisteredHitCounters::validateEveryN(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
01837     base::threading::ScopedLock scopedLock(lock());
01838     base::HitCounter* counter = get(filename, lineNumber);
01839     if (counter == nullptr) {
01840         registerNew(counter = new base::HitCounter(filename, lineNumber));
01841     }
01842     counter->validateHitCounts(n);
01843     bool result = (n >= 1 && counter->hitCounts() != 0 && counter->hitCounts() % n == 0);
01844     return result;
01845 }
01846
01849 bool RegisteredHitCounters::validateAfterN(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
01850     base::threading::ScopedLock scopedLock(lock());
01851     base::HitCounter* counter = get(filename, lineNumber);
01852     if (counter == nullptr) {
01853         registerNew(counter = new base::HitCounter(filename, lineNumber));
01854     }
01855     // Do not use validateHitCounts here since we do not want to reset counter here
01856     // Note the >= instead of > because we are incrementing
01857     // after this check
01858     if (counter->hitCounts() >= n)
01859         return true;
01860     counter->increment();
01861     return false;
01862 }
01863
01866 bool RegisteredHitCounters::validateNTimes(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
01867     base::threading::ScopedLock scopedLock(lock());
01868     base::HitCounter* counter = get(filename, lineNumber);
01869     if (counter == nullptr) {
01870         registerNew(counter = new base::HitCounter(filename, lineNumber));
01871     }
01872     counter->increment();
01873     // Do not use validateHitCounts here since we do not want to reset counter here
01874     if (counter->hitCounts() <= n)
01875         return true;
01876     return false;
01877 }
01878
01879 // RegisteredLoggers
01880
01881 RegisteredLoggers::RegisteredLoggers(const LogBuilderPtr& defaultLogBuilder) :
01882     m_defaultLogBuilder(defaultLogBuilder) {
01883     m_defaultConfigurations.setToDefault();
01884     m_logStreamsReference = std::make_shared<base::LogStreamsReferenceMap>();
01885 }
01886
01887 Logger* RegisteredLoggers::get(const std::string& id, bool forceCreation) {
01888     base::threading::ScopedLock scopedLock(lock());
01889     Logger* logger_ = base::utils::Registry<Logger, std::string>::get(id);
01890     if (logger_ == nullptr && forceCreation) {
01891         bool validId = Logger::isValidId(id);
01892         if (!validId) {
01893             ELPP_ASSERT(validId, "Invalid logger ID [" << id << "]. Not registering this logger.");
01894             return nullptr;
01895         }
01896         logger_ = new Logger(id, m_defaultConfigurations, m_logStreamsReference);
01897         logger_->m_logBuilder = m_defaultLogBuilder;
01898         registerNew(id, logger_);
01899         LoggerRegistrationCallback* callback = nullptr;
01900         for (const std::pair<std::string, base::type::LoggerRegistrationCallbackPtr>& h
: m_loggerRegistrationCallbacks) {
01901             callback = h.second.get();
01902             if (callback != nullptr && callback->enabled()) {
01903                 callback->handle(logger_);
01904             }
01905         }
01906     }
01907     return logger_;
01908 }
01909
01910
01911 bool RegisteredLoggers::remove(const std::string& id) {
01912     if (id == base::consts::kDefaultLoggerId) {
01913         return false;
01914     }
01915     // get has internal lock
01916     Logger* logger = base::utils::Registry<Logger, std::string>::get(id);
01917     if (logger != nullptr) {
01918         // unregister has internal lock
01919         unregister(logger);
01920     }
01921     return true;
01922 }

```

```

01923
01924 void RegisteredLoggers::unsafeFlushAll(void) {
01925     ELPP_INTERNAL_INFO(1, "Flushing all log files");
01926     for (base::LogStreamsReferenceMap::iterator it = m_logStreamsReference->begin();
01927          it != m_logStreamsReference->end(); ++it) {
01928         if (it->second.get() == nullptr) continue;
01929         it->second->flush();
01930     }
01931 }
01932
01933 // VRegistry
01934
01935 VRegistry::VRegistry(base::type::VerboseLevel level, base::type::EnumType* pFlags) : m_level(level),
01936     m_pFlags(pFlags) {
01937
01938 void VRegistry::setLevel(base::type::VerboseLevel level) {
01939     base::threading::ScopedLock scopedLock(lock());
01940     if (level > 9)
01941         m_level = base::consts::kMaxVerboseLevel;
01942     else
01943         m_level = level;
01944 }
01945
01946 void VRegistry::setModules(const char* modules) {
01947     base::threading::ScopedLock scopedLock(lock());
01948     auto addSuffix = [](std::stringstream& ss, const char* sfx, const char* prev) {
01949         if (prev != nullptr && base::utils::Str::endsWith(ss.str(), std::string(prev))) {
01950             std::string chr(ss.str().substr(0, ss.str().size() - strlen(prev)));
01951             ss.str(std::string(""));
01952             ss << chr;
01953         }
01954         if (base::utils::Str::endsWith(ss.str(), std::string(sfx))) {
01955             std::string chr(ss.str().substr(0, ss.str().size() - strlen(sfx)));
01956             ss.str(std::string(""));
01957             ss << chr;
01958         }
01959         ss << sfx;
01960     };
01961
01962     auto insert = [&](std::stringstream& ss, base::type::VerboseLevel level) {
01963         if (!base::utils::hasFlag(LoggingFlag::DisableVModulesExtensions, *m_pFlags)) {
01964             addSuffix(ss, ".h", nullptr);
01965             m_modules.insert(std::make_pair(ss.str(), level));
01966             addSuffix(ss, ".c", ".h");
01967             m_modules.insert(std::make_pair(ss.str(), level));
01968             addSuffix(ss, ".cpp", ".c");
01969             m_modules.insert(std::make_pair(ss.str(), level));
01970             addSuffix(ss, ".cc", ".cpp");
01971             m_modules.insert(std::make_pair(ss.str(), level));
01972             addSuffix(ss, ".cxx", ".cc");
01973             m_modules.insert(std::make_pair(ss.str(), level));
01974             addSuffix(ss, ".-inl.h", ".cxx");
01975             m_modules.insert(std::make_pair(ss.str(), level));
01976             addSuffix(ss, ".hxx", ".-inl.h");
01977             m_modules.insert(std::make_pair(ss.str(), level));
01978             addSuffix(ss, ".hpp", ".hxx");
01979             m_modules.insert(std::make_pair(ss.str(), level));
01980             addSuffix(ss, ".hh", ".hpp");
01981         }
01982         m_modules.insert(std::make_pair(ss.str(), level));
01983     };
01984     bool isMod = true;
01985     bool isLevel = false;
01986     std::stringstream ss;
01987     int level = -1;
01988     for (; *modules; ++modules) {
01989         switch (*modules) {
01990             case '=':
01991                 isLevel = true;
01992                 isMod = false;
01993                 break;
01994             case ',':
01995                 isLevel = false;
01996                 isMod = true;
01997                 if (!ss.str().empty() && level != -1) {
01998                     insert(ss, static_cast<base::type::VerboseLevel>(level));
01999                     ss.str(std::string(""));
02000                     level = -1;
02001                 }
02002                 break;
02003             default:
02004                 if (isMod) {
02005                     ss << *modules;
02006                 } else if (isLevel) {
02007                     if (isdigit(*modules)) {
02008                         level = static_cast<base::type::VerboseLevel>(*modules) - 48;
02009                     }

```

```

02010     }
02011     break;
02012 }
02013 }
02014 if (!ss.str().empty() && level != -1) {
02015     insert(ss, static_cast<base::type::VerboseLevel>(level));
02016 }
02017 }
02018
02019 bool VRegistry::allowed(base::type::VerboseLevel vlevel, const char* file) {
02020     base::threading::ScopedLock scopedLock(lock());
02021     if (m_modules.empty() || file == nullptr) {
02022         return vlevel <= m_level;
02023     } else {
02024         char baseFilename[base::consts::kSourceFilenameMaxLength] = "";
02025         base::utils::File::buildBaseFilename(file, baseFilename);
02026         std::unordered_map<std::string, base::type::VerboseLevel>::iterator it = m_modules.begin();
02027         for (; it != m_modules.end(); ++it) {
02028             if (base::utils::Str::wildCardMatch(baseFilename, it->first.c_str())) {
02029                 return vlevel <= it->second;
02030             }
02031         }
02032         if (base::utils::hasFlag(LoggingFlag::AllowVerboseIfModuleNotSpecified, *m_pFlags)) {
02033             return true;
02034         }
02035         return false;
02036     }
02037 }
02038
02039 void VRegistry::setFromArgs(const base::utils::CommandLineArgs* commandLineArgs) {
02040     if (commandLineArgs->hasParam("-v") || commandLineArgs->hasParam("--verbose") ||
02041         commandLineArgs->hasParam("-V") || commandLineArgs->hasParam("--VERBOSE")) {
02042         setLevel(base::consts::kMaxVerboseLevel);
02043     } else if (commandLineArgs->hasParamWithValue("--v")) {
02044         setLevel(static_cast<base::type::VerboseLevel>(atoi(commandLineArgs->getParamValue("--v"))));
02045     } else if (commandLineArgs->hasParamWithValue("--V")) {
02046         setLevel(static_cast<base::type::VerboseLevel>(atoi(commandLineArgs->getParamValue("--V"))));
02047     } else if ((commandLineArgs->hasParamWithValue("-vmodule") && vModulesEnabled()) {
02048         setModules(commandLineArgs->getParamValue("-vmodule"));
02049     } else if (commandLineArgs->hasParamWithValue("-VMODULE") && vModulesEnabled()) {
02050         setModules(commandLineArgs->getParamValue("-VMODULE"));
02051     }
02052 }
02053
02054 #if !defined(ELPP_DEFAULT_LOGGING_FLAGS)
02055 #   define ELPP_DEFAULT_LOGGING_FLAGS 0x0
02056 #endif // !defined(ELPP_DEFAULT_LOGGING_FLAGS)
02057 // Storage
02058 #if ELPP_ASYNC_LOGGING
02059 Storage::Storage(const LogBuilderPtr& defaultLogBuilder, base::IWorker* asyncDispatchWorker) :
02060 #else
02061 Storage::Storage(const LogBuilderPtr& defaultLogBuilder) :
02062 #endif // ELPP_ASYNC_LOGGING
02063     m_registeredHitCounters(new base::RegisteredHitCounters()),
02064     m_registeredLoggers(new base::RegisteredLoggers(defaultLogBuilder)),
02065     m_flags(ELPP_DEFAULT_LOGGING_FLAGS),
02066     m_vRegistry(new base::VRegistry(0, &m_flags)),
02067
02068 #if ELPP_ASYNC_LOGGING
02069     m_asyncLogQueue(new base::AsyncLogQueue()),
02070     m_asyncDispatchWorker(asyncDispatchWorker),
02071 #endif // ELPP_ASYNC_LOGGING
02072
02073     m_preRollOutCallback(base::defaultPreRollOutCallback) {
02074         // Register default logger
02075         m_registeredLoggers->get(std::string(base::consts::kDefaultLoggerId));
02076         // We register default logger anyway (worse case it's not going to register) just in case
02077         m_registeredLoggers->get("default");
02078
02079 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02080         // Register performance logger and reconfigure format
02081         Logger* performanceLogger =
02082             m_registeredLoggers->get(std::string(base::consts::kPerformanceLoggerId));
02083         m_registeredLoggers->get("performance");
02084         performanceLogger->configurations()->setGlobally(ConfigurationType::Format, std::string("%datetime
02085 %level %msg"));
02086         performanceLogger->reconfigure();
02087 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02088
02089 #if defined(ELPP_SYSLOG)
02090         // Register syslog logger and reconfigure format
02091         Logger* syslogLogger = m_registeredLoggers->get(std::string(base::consts::kSysLogLoggerId));
02092         syslogLogger->configurations()->setGlobally(ConfigurationType::Format, std::string("%level: %msg"));
02093         syslogLogger->reconfigure();
02094 #endif // defined(ELPP_SYSLOG)
02095         addFlag(LoggingFlag::AllowVerboseIfModuleNotSpecified);
02096 #if ELPP_ASYNC_LOGGING

```



```

02095     installLogDispatchCallback<base::AsyncLogDispatchCallback>(std::string("AsyncLogDispatchCallback"));
02096 #else
02097     installLogDispatchCallback<base::DefaultLogDispatchCallback>(std::string("DefaultLogDispatchCallback"));
02098 #endif // ELPP_ASYNC_LOGGING
02099 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02100     installPerformanceTrackingCallback<base::DefaultPerformanceTrackingCallback>
02101     (std::string("DefaultPerformanceTrackingCallback"));
02102 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02103     ELPP_INTERNAL_INFO(1, "Easylogging++ has been initialized");
02104 #if ELPP_ASYNC_LOGGING
02105     m_asyncDispatchWorker->start();
02106 #endif // ELPP_ASYNC_LOGGING
02107 }
02108
02109 Storage::~Storage(void) {
02110     ELPP_INTERNAL_INFO(4, "Destroying storage");
02111 #if ELPP_ASYNC_LOGGING
02112     ELPP_INTERNAL_INFO(5, "Replacing log dispatch callback to synchronous");
02113
02114     uninstallLogDispatchCallback<base::AsyncLogDispatchCallback>(std::string("AsyncLogDispatchCallback"));
02115
02116     installLogDispatchCallback<base::DefaultLogDispatchCallback>(std::string("DefaultLogDispatchCallback"));
02117     ELPP_INTERNAL_INFO(5, "Destroying asyncDispatchWorker");
02118     base::utils::safeDelete(m_asyncDispatchWorker);
02119     ELPP_INTERNAL_INFO(5, "Destroying asyncLogQueue");
02120     base::utils::safeDelete(m_asyncLogQueue);
02121 #endif // ELPP_ASYNC_LOGGING
02122     ELPP_INTERNAL_INFO(5, "Destroying registeredHitCounters");
02123     base::utils::safeDelete(m_registeredHitCounters);
02124     ELPP_INTERNAL_INFO(5, "Destroying registeredLoggers");
02125     base::utils::safeDelete(m_registeredLoggers);
02126     ELPP_INTERNAL_INFO(5, "Destroying vRegistry");
02127     base::utils::safeDelete(m_vRegistry);
02128 }
02129
02130 bool Storage::hasCustomFormatSpecifier(const char* formatSpecifier) {
02131     base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
02132     return std::find(m_customFormatSpecifiers.begin(), m_customFormatSpecifiers.end(),
02133         formatSpecifier) != m_customFormatSpecifiers.end();
02134 }
02135
02136 void Storage::installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier) {
02137     if (hasCustomFormatSpecifier(customFormatSpecifier.formatSpecifier())) {
02138         return;
02139     }
02140     base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
02141     m_customFormatSpecifiers.push_back(customFormatSpecifier);
02142 }
02143
02144 bool Storage::uninstallCustomFormatSpecifier(const char* formatSpecifier) {
02145     base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
02146     std::vector<CustomFormatSpecifier>::iterator it = std::find(m_customFormatSpecifiers.begin(),
02147         m_customFormatSpecifiers.end(), formatSpecifier);
02148     if (it != m_customFormatSpecifiers.end() && strcmp(formatSpecifier, it->formatSpecifier()) == 0) {
02149         m_customFormatSpecifiers.erase(it);
02150         return true;
02151     }
02152     return false;
02153 }
02154
02155 void Storage::setApplicationArguments(int argc, char** argv) {
02156     m_commandLineArgs.setArgs(argc, argv);
02157     m_vRegistry->setFromArgs(commandLineArgs());
02158     // default log file
02159 #if !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
02160     if (m_commandLineArgs.hasParamWithValue(base::consts::kDefaultLogFileParam)) {
02161         Configurations c;
02162         c.setGlobally(ConfigurationType::Filename,
02163             std::string(m_commandLineArgs.getParamValue(base::consts::kDefaultLogFileParam)));
02164         registeredLoggers()->setDefaultConfigurations(c);
02165         for (base::RegisteredLoggers::iterator it = registeredLoggers()->begin();
02166             it != registeredLoggers()->end(); ++it) {
02167             it->second->configure(c);
02168         }
02169     }
02170 #endif // !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
02171 #if defined(ELPP_LOGGING_FLAGS_FROM_ARG)
02172     if (m_commandLineArgs.hasParamWithValue(base::consts::kLoggingFlagsParam)) {
02173         int userInput = atoi(m_commandLineArgs.getParamValue(base::consts::kLoggingFlagsParam));
02174         if (ELPP_DEFAULT_LOGGING_FLAGS == 0x0) {
02175             m_flags = userInput;
02176         } else {
02177             base::utils::addFlag<base::type::EnumType>(userInput, &m_flags);
02178         }
02179     }
02180 #endif // defined(ELPP_LOGGING_FLAGS_FROM_ARG)

```



```

02179 }
02180
02181 } // namespace base
02182
02183 // LogDispatchCallback
02184 #if defined(ELPP_THREAD_SAFE)
02185 void LogDispatchCallback::handle(const LogDispatchData* data) {
02186     base::threading::ScopedLock scopedLock(m_fileLocksMapLock);
02187     std::string filename =
02188     data->logMessage()->logger()->typedConfigurations()->filename(data->logMessage()->level());
02189     auto lock = m_fileLocks.find(filename);
02190     if (lock == m_fileLocks.end()) {
02191         m_fileLocks.emplace(std::make_pair(filename, std::unique_ptr<base::threading::Mutex>(new
02192         base::threading::Mutex)));
02193     }
02194 }
02195 #else
02196 void LogDispatchCallback::handle(const LogDispatchData* /*data*/) {}
02197 #endif
02198
02199 base::threading::Mutex& LogDispatchCallback::fileHandle(const LogDispatchData* data) {
02200     auto it =
02201     m_fileLocks.find(data->logMessage()->logger()->typedConfigurations()->filename(data->logMessage()->level()));
02202     return *(it->second.get());
02203 }
02204
02205 namespace base {
02206 // DefaultLogDispatchCallback
02207 void DefaultLogDispatchCallback::handle(const LogDispatchData* data) {
02208     #if defined(ELPP_THREAD_SAFE)
02209     LogDispatchCallback::handle(data);
02210     base::threading::ScopedLock scopedLock(fileHandle(data));
02211     #endif
02212     m_data = data;
02213     dispatch(m_data->logMessage()->logger()->logBuilder()->build(m_data->logMessage(),
02214     m_data->dispatchAction() == base::DispatchAction::NormalLog));
02215 }
02216
02217 void DefaultLogDispatchCallback::dispatch(base::type::string_t&& logLine) {
02218     if (m_data->dispatchAction() == base::DispatchAction::NormalLog) {
02219         if (m_data->logMessage()->logger()->m_typedConfigurations->toFile(m_data->logMessage()->level()))
02220         {
02221             base::type::fstream_t* fs = m_data->logMessage()->logger()->m_typedConfigurations->fileStream(
02222             m_data->logMessage()->level());
02223             if (fs != nullptr) {
02224                 fs->write(logLine.c_str(), logLine.size());
02225                 if (fs->fail()) {
02226                     ELPP_INTERNAL_ERROR("Unable to write log to file ["
02227                     «
02228                     m_data->logMessage()->logger()->m_typedConfigurations->filename(m_data->logMessage()->level()) «
02229                     "].\n"
02230                     « "Few possible reasons (could be something else):\n" « " *
02231                     Permission denied\n"
02232                     « " * Disk full\n" « " * Disk is not writable", true);
02233                 } else {
02234                     if (ELPP->hasFlag(LoggingFlag::ImmediateFlush)
02235                     || (m_data->logMessage()->logger()->isFlushNeeded(m_data->logMessage()->level())) {
02236                         m_data->logMessage()->logger()->flush(m_data->logMessage()->level(), fs);
02237                     }
02238                 }
02239             } else {
02240                 ELPP_INTERNAL_ERROR("Log file for [" «
02241                 LevelHelper::convertToString(m_data->logMessage()->level()) « "]" "
02242                 « "has not been configured but [TO_FILE] is configured to TRUE. [Logger
02243                 ID: "
02244                 « m_data->logMessage()->logger()->id() « "]", false);
02245             }
02246         }
02247         if (m_data->logMessage()->logger()->m_typedConfigurations->toStandardOutput(m_data->logMessage()->level()))
02248         {
02249             if (ELPP->hasFlag(LoggingFlag::ColoredTerminalOutput))
02250             m_data->logMessage()->logger()->logBuilder()->convertToColoredOutput(&logLine,
02251             m_data->logMessage()->level());
02252             ELPP_COUT « ELPP_COUT_LINE(logLine);
02253         }
02254     }
02255     #if defined(ELPP_SYSLOG)
02256     else if (m_data->dispatchAction() == base::DispatchAction::SysLog) {
02257         // Determine syslog priority
02258         int syslogPriority = 0;
02259         if (m_data->logMessage()->level() == Level::Fatal)
02260             syslogPriority = LOG_EMERG;
02261         else if (m_data->logMessage()->level() == Level::Error)
02262             syslogPriority = LOG_ERR;
02263         else if (m_data->logMessage()->level() == Level::Warning)

```

```

02254     syslogPriority = LOG_WARNING;
02255     else if (m_data->logMessage()->level() == Level::Info)
02256         syslogPriority = LOG_INFO;
02257     else if (m_data->logMessage()->level() == Level::Debug)
02258         syslogPriority = LOG_DEBUG;
02259     else
02260         syslogPriority = LOG_NOTICE;
02261 # if defined(ELPP_UNICODE)
02262     char* line = base::utils::Str::wcharPtrToCharPtr(logLine.c_str());
02263     syslog(sysLogPriority, "%s", line);
02264     free(line);
02265 # else
02266     syslog(sysLogPriority, "%s", logLine.c_str());
02267 # endif
02268 }
02269 #endif // defined(ELPP_SYSLOG)
02270 }
02271
02272 #if ELPP_ASYNC_LOGGING
02273
02274 // AsyncLogDispatchCallback
02275
02276 void AsyncLogDispatchCallback::handle(const LogDispatchData* data) {
02277     base::type::string_t logLine = data->logMessage()->logger()->logBuilder()->build(data->logMessage(),
02278     data->dispatchAction() == base::DispatchAction::NormalLog);
02279     if (data->dispatchAction() == base::DispatchAction::NormalLog
02280     &&
02281     data->logMessage()->logger()->typedConfigurations()->toStandardOutput(data->logMessage()->level())) {
02282         if (ELPP->hasFlag(LoggingFlag::ColoredTerminalOutput))
02283             data->logMessage()->logger()->logBuilder()->convertToColoredOutput(&logLine,
02284             data->logMessage()->level());
02285         ELPP_COUT << ELPP_COUT_LINE(logLine);
02286     }
02287     // Save resources and only queue if we want to write to file otherwise just ignore handler
02288     if (data->logMessage()->logger()->typedConfigurations()->toFile(data->logMessage()->level())) {
02289         ELPP->asyncLogQueue()->push(AsyncLogItem(*(data->logMessage()), *data, logLine));
02290     }
02291 }
02292
02293 // AsyncDispatchWorker
02294 AsyncDispatchWorker::AsyncDispatchWorker() {
02295     setContinueRunning(false);
02296 }
02297
02298 AsyncDispatchWorker::~AsyncDispatchWorker() {
02299     setContinueRunning(false);
02300     ELPP_INTERNAL_INFO(6, "Stopping dispatch worker - Cleaning log queue");
02301     clean();
02302     ELPP_INTERNAL_INFO(6, "Log queue cleaned");
02303 }
02304
02305 bool AsyncDispatchWorker::clean(void) {
02306     std::mutex m;
02307     std::unique_lock<std::mutex> lk(m);
02308     cv.wait(lk, [] { return !ELPP->asyncLogQueue()->empty(); });
02309     emptyQueue();
02310     lk.unlock();
02311     cv.notify_one();
02312     return ELPP->asyncLogQueue()->empty();
02313 }
02314
02315 void AsyncDispatchWorker::emptyQueue(void) {
02316     while (!ELPP->asyncLogQueue()->empty()) {
02317         AsyncLogItem data = ELPP->asyncLogQueue()->next();
02318         handle(&data);
02319         base::threading::msleep(100);
02320     }
02321 }
02322
02323 void AsyncDispatchWorker::start(void) {
02324     base::threading::msleep(5000); // 5s (why?)
02325     setContinueRunning(true);
02326     std::thread t1(&AsyncDispatchWorker::run, this);
02327     t1.join();
02328 }
02329
02330 void AsyncDispatchWorker::handle(AsyncLogItem* logItem) {
02331     LogDispatchData* data = logItem->data();
02332     LogMessage* logMessage = logItem->logMessage();
02333     Logger* logger = logMessage->logger();
02334     base::TypedConfigurations* conf = logger->typedConfigurations();
02335     base::type::string_t logLine = logItem->logLine();
02336     if (data->dispatchAction() == base::DispatchAction::NormalLog) {
02337         if (conf->toFile(logMessage->level())) {
02338             base::type::fstream_t* fs = conf->fileStream(logMessage->level());
02339             if (fs != nullptr) {
02340                 fs->write(logLine.c_str(), logLine.size());
02341             }
02342         }
02343     }
02344 }

```

```

02339         if (fs->fail()) {
02340             ELPP_INTERNAL_ERROR("Unable to write log to file ["
02341                                 « conf->filename(logMessage->level()) « "].\n"
02342                                 « "Few possible reasons (could be something else):\n" « "      *
Permission denied\n"
02343                                 « "      * Disk full\n" « "      * Disk is not writable", true);
02344         } else {
02345             if (ELPP->hasFlag(LoggingFlag::ImmediateFlush) ||
(logger->isFlushNeeded(logMessage->level())) {
02346                 logger->flush(logMessage->level(), fs);
02347             }
02348         }
02349     } else {
02350         ELPP_INTERNAL_ERROR("Log file for [" « LevelHelper::convertToString(logMessage->level()) « "]
"
02351                             « "has not been configured but [TO_FILE] is configured to TRUE. [Logger
ID: " « logger->id() « "]", false);
02352     }
02353 }
02354 }
02355 # if defined(ELPP_SYSLOG)
02356 else if (data->dispatchAction() == base::DispatchAction::SysLog) {
02357     // Determine syslog priority
02358     int syslogPriority = 0;
02359     if (logMessage->level() == Level::Fatal)
02360         syslogPriority = LOG_EMERG;
02361     else if (logMessage->level() == Level::Error)
02362         syslogPriority = LOG_ERR;
02363     else if (logMessage->level() == Level::Warning)
02364         syslogPriority = LOG_WARNING;
02365     else if (logMessage->level() == Level::Info)
02366         syslogPriority = LOG_INFO;
02367     else if (logMessage->level() == Level::Debug)
02368         syslogPriority = LOG_DEBUG;
02369     else
02370         syslogPriority = LOG_NOTICE;
02371 # if defined(ELPP_UNICODE)
02372     char* line = base::utils::Str::wcharPtrToCharPtr(logLine.c_str());
02373     syslog(syslogPriority, "%s", line);
02374     free(line);
02375 # else
02376     syslog(syslogPriority, "%s", logLine.c_str());
02377 # endif
02378 }
02379 # endif // defined(ELPP_SYSLOG)
02380 }
02381 void AsyncDispatchWorker::run(void) {
02382     while (continueRunning()) {
02383         emptyQueue();
02384         base::threading::msleep(10); // 10ms
02385     }
02386 }
02387 }
02388 #endif // ELPP_ASYNC_LOGGING
02389
02390 // DefaultLogBuilder
02391
02392 base::type::string_t DefaultLogBuilder::build(const LogMessage* logMessage, bool appendNewLine) const
02393 {
02394     base::TypedConfigurations* tc = logMessage->logger()->typedConfigurations();
02395     const base::LogFormat* logFormat = &tc->logFormat(logMessage->level());
02396     base::type::string_t logLine = logFormat->format();
02397     char buff[base::consts::kSourceFilenameMaxLength + base::consts::kSourceLineMaxLength] = "";
02398     const char* bufLim = buff + sizeof(buff);
02399     if (logFormat->hasFlag(base::FormatFlags::AppName)) {
02400         // App name
02401         base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kAppNameFormatSpecifier,
logMessage->logger()->parentApplicationName());
02402     }
02403     if (logFormat->hasFlag(base::FormatFlags::ThreadId)) {
02404         // Thread ID
02405         base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kThreadIdFormatSpecifier,
ELPP->getThreadName(base::threading::getCurrentThreadId()));
02406     }
02407     if (logFormat->hasFlag(base::FormatFlags::DateTime)) {
02408         // DateTime
02409         base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kDateTimeFormatSpecifier,
base::utils::DateTime::getDateTime(logFormat->dateTimeFormat().c_str(),
&tc->subsecondPrecision(logMessage->level())));
02410     }
02411     if (logFormat->hasFlag(base::FormatFlags::Function)) {
02412         // Function
02413         base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFunctionFormatSpecifier,
logMessage->func());
02414     }
02415     if (logFormat->hasFlag(base::FormatFlags::File)) {
02416         // File

```

```

02420     base::utils::Str::clearBuff(buff, base::consts::kSourceFilenameMaxLength);
02421     base::utils::File::buildStrippedFilename(logMessage->file().c_str(), buff);
02422     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFileFormatSpecifier,
std::string(buff));
02423 }
02424 if (logFormat->hasFlag(base::FormatFlags::FileBase)) {
02425     // FileBase
02426     base::utils::Str::clearBuff(buff, base::consts::kSourceFilenameMaxLength);
02427     base::utils::File::buildBaseFilename(logMessage->file(), buff);
02428     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFileBaseFormatSpecifier,
std::string(buff));
02429 }
02430 if (logFormat->hasFlag(base::FormatFlags::Line)) {
02431     // Line
02432     char* buf = base::utils::Str::clearBuff(buff, base::consts::kSourceLineMaxLength);
02433     buf = base::utils::Str::convertAndAddToBuff(logMessage->line(),
base::consts::kSourceLineMaxLength, buf, bufLim, false);
02434     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogLineFormatSpecifier,
std::string(buff));
02435 }
02436 if (logFormat->hasFlag(base::FormatFlags::Location)) {
02437     // Location
02438     char* buf = base::utils::Str::clearBuff(buff,
base::consts::kSourceFilenameMaxLength +
base::consts::kSourceLineMaxLength);
02440     base::utils::File::buildStrippedFilename(logMessage->file().c_str(), buff);
02441     buf = base::utils::Str::addToBuff(buff, buf, bufLim);
02442     buf = base::utils::Str::addToBuff(":", buf, bufLim);
02443     buf = base::utils::Str::convertAndAddToBuff(logMessage->line(),
base::consts::kSourceLineMaxLength, buf, bufLim,
false);
02445     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogLocationFormatSpecifier,
std::string(buff));
02446 }
02447 if (logMessage->level() == Level::Verbose && logFormat->hasFlag(base::FormatFlags::VerboseLevel)) {
02448     // Verbose level
02449     char* buf = base::utils::Str::clearBuff(buff, 1);
02450     buf = base::utils::Str::convertAndAddToBuff(logMessage->verboseLevel(), 1, buf, bufLim, false);
02451     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kVerboseLevelFormatSpecifier,
std::string(buff));
02452 }
02453 if (logFormat->hasFlag(base::FormatFlags::LogMessage)) {
02454     // Log message
02455     base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kMessageFormatSpecifier,
logMessage->message());
02456 }
02457 #if !defined(ELPP_DISABLE_CUSTOM_FORMAT_SPECIFIERS)
02458 el::base::threading::ScopedLock lock_(ELPP->customFormatSpecifiersLock());
02459 ELPP_UNUSED(lock_);
02460 for (std::vector<CustomFormatSpecifier>::const_iterator it =
ELPP->customFormatSpecifiers()->begin();
02461      it != ELPP->customFormatSpecifiers()->end(); ++it) {
02462     std::string fs(it->formatSpecifier());
02463     base::type::string_t wcsFormatSpecifier(fs.begin(), fs.end());
02464     base::utils::Str::replaceFirstWithEscape(logLine, wcsFormatSpecifier, it->resolver()(logMessage));
02465 }
02466 #endif // !defined(ELPP_DISABLE_CUSTOM_FORMAT_SPECIFIERS)
02467 if (appendNewLine) logLine += ELPP_LITERAL("\n");
02468 return logLine;
02469 }
02470
02471 // LogDispatcher
02472
02473 void LogDispatcher::dispatch(void) {
02474     if (m_proceed && m_dispatchAction == base::DispatchAction::None) {
02475         m_proceed = false;
02476     }
02477     if (!m_proceed) {
02478         return;
02479     }
02480 #ifndef ELPP_NO_GLOBAL_LOCK
02481     // see https://github.com/muflihun/easyloggingpp/issues/580
02482     // global lock is turned on by default unless
02483     // ELPP_NO_GLOBAL_LOCK is defined
02484     base::threading::ScopedLock scopedLock(ELPP->lock());
02485 #endif
02486     base::TypedConfigurations* tc = m_logMessage->logger()->m_typedConfigurations;
02487     if (ELPP->hasFlag(LoggingFlag::StrictLogFileSizeCheck)) {
02488         tc->validateFileRolling(m_logMessage->level(), ELPP->preRollOutCallback());
02489     }
02490     LogDispatchCallback* callback = nullptr;
02491     LogDispatchData data;
02492     for (const std::pair<std::string, base::type::LogDispatchCallbackPtr>& h
: ELPP->m_logDispatchCallbacks) {
02493         callback = h.second.get();
02494         if (callback != nullptr && callback->enabled()) {
02495             data.setLogMessage(m_logMessage);

```

```

02497         data.setDispatchAction(m_dispatchAction);
02498         callback->handle(&data);
02499     }
02500 }
02501 }
02502
02503 // MessageBuilder
02504
02505 void MessageBuilder::initialize(Logger* logger) {
02506     m_logger = logger;
02507     m_containerLogSeparator = ELPP->hasFlag(LoggingFlag::NewLineForContainer) ?
02508         ELPP_LITERAL("\n    ") : ELPP_LITERAL(", ");
02509 }
02510
02511 MessageBuilder& MessageBuilder::operator«(const wchar_t* msg) {
02512     if (msg == nullptr) {
02513         m_logger->stream() << base::consts::kNullPointer;
02514         return *this;
02515     }
02516     # if defined(ELPP_UNICODE)
02517     m_logger->stream() << msg;
02518     # else
02519     char* buff_ = base::utils::Str::wcharPtrToCharPtr(msg);
02520     m_logger->stream() << buff_;
02521     free(buff_);
02522     # endif
02523     if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {
02524         m_logger->stream() << " ";
02525     }
02526     return *this;
02527 }
02528
02529 // Writer
02530
02531 Writer& Writer::construct(Logger* logger, bool needLock) {
02532     m_logger = logger;
02533     initializeLogger(logger->id(), false, needLock);
02534     m_messageBuilder.initialize(m_logger);
02535     return *this;
02536 }
02537
02538 Writer& Writer::construct(int count, const char* loggerIds, ...) {
02539     if (ELPP->hasFlag(LoggingFlag::MultiLoggerSupport)) {
02540         va_list loggersList;
02541         va_start(loggersList, loggerIds);
02542         const char* id = loggerIds;
02543         m_loggerIds.reserve(count);
02544         for (int i = 0; i < count; ++i) {
02545             m_loggerIds.push_back(std::string(id));
02546             id = va_arg(loggersList, const char*);
02547         }
02548         va_end(loggersList);
02549         initializeLogger(m_loggerIds.at(0));
02550     } else {
02551         initializeLogger(std::string(loggerIds));
02552     }
02553     m_messageBuilder.initialize(m_logger);
02554     return *this;
02555 }
02556
02557 void Writer::initializeLogger(const std::string& loggerId, bool lookup, bool needLock) {
02558     if (lookup) {
02559         m_logger = ELPP->registeredLoggers()->get(loggerId,
02560             ELPP->hasFlag(LoggingFlag::CreateLoggerAutomatically));
02561     }
02562     if (m_logger == nullptr) {
02563         if (!ELPP->registeredLoggers()->has(std::string(base::consts::kDefaultLoggerId))) {
02564             // Somehow default logger has been unregistered. Not good! Register again
02565             ELPP->registeredLoggers()->get(std::string(base::consts::kDefaultLoggerId));
02566         }
02567     }
02568     Writer(Level::Debug, m_file, m_line, m_func).construct(1, base::consts::kDefaultLoggerId)
02569         << "Logger [" << loggerId << "] is not registered yet!";
02570     m_proceed = false;
02571 } else {
02572     if (needLock) {
02573         m_logger->acquireLock(); // This should not be unlocked by checking m_proceed because
02574         // m_proceed can be changed by lines below
02575     }
02576     if (ELPP->hasFlag(LoggingFlag::HierarchicalLogging)) {
02577         m_proceed = m_level == Level::Verbose ? m_logger->enabled(m_level) :
02578             LevelHelper::castToInt(m_level) >= LevelHelper::castToInt(ELPP->m_loggingLevel);
02579     } else {
02580         m_proceed = m_logger->enabled(m_level);
02581     }
02582 }

```

```

02583 }
02584
02585 void Writer::processDispatch() {
02586 #if ELPP_LOGGING_ENABLED
02587     if (ELPP->hasFlag(LoggingFlag::MultiLoggerSupport)) {
02588         bool firstDispatched = false;
02589         base::type::string_t logMessage;
02590         std::size_t i = 0;
02591         do {
02592             if (m_proceed) {
02593                 if (firstDispatched) {
02594                     m_logger->stream() << logMessage;
02595                 } else {
02596                     firstDispatched = true;
02597                     if (m_loggerIds.size() > 1) {
02598                         logMessage = m_logger->stream().str();
02599                     }
02600                 }
02601                 triggerDispatch();
02602             } else if (m_logger != nullptr) {
02603                 m_logger->stream().str(ELPP_LITERAL(""));
02604                 m_logger->releaseLock();
02605             }
02606             if (i + 1 < m_loggerIds.size()) {
02607                 initializeLogger(m_loggerIds.at(i + 1));
02608             }
02609             while (++i < m_loggerIds.size());
02610         } else {
02611             if (m_proceed) {
02612                 triggerDispatch();
02613             } else if (m_logger != nullptr) {
02614                 m_logger->stream().str(ELPP_LITERAL(""));
02615                 m_logger->releaseLock();
02616             }
02617         }
02618     } else
02619     if (m_logger != nullptr) {
02620         m_logger->stream().str(ELPP_LITERAL(""));
02621         m_logger->releaseLock();
02622     }
02623 #endif // ELPP_LOGGING_ENABLED
02624 }
02625
02626 void Writer::triggerDispatch(void) {
02627     try {
02628         if (m_proceed) {
02629             if (m_msg == nullptr) {
02630                 LogMessage msg(m_level, m_file, m_line, m_func, m_verboseLevel,
02631                     m_logger);
02632                 base::LogDispatcher(m_proceed, &msg, m_dispatchAction).dispatch();
02633             } else {
02634                 base::LogDispatcher(m_proceed, m_msg, m_dispatchAction).dispatch();
02635             }
02636         }
02637         if (m_logger != nullptr) {
02638             m_logger->stream().str(ELPP_LITERAL(""));
02639             m_logger->releaseLock();
02640         }
02641         if (m_proceed && m_level == Level::Fatal
02642             && !ELPP->hasFlag(LoggingFlag::DisableApplicationAbortOnFatalLog)) {
02643             base::Writer(Level::Warning, m_file, m_line, m_func).construct(1,
02644                 base::consts::kDefaultLoggerId)
02645                 << "Aborting application. Reason: Fatal log at [" << m_file << ":" << m_line << "];"
02646                 << std::stringstream reasonStream;
02647                 reasonStream << "Fatal log at [" << m_file << ":" << m_line << "]"
02648                 << " If you wish to disable 'abort on fatal log' please use "
02649                 << "el::Loggers::addFlag(el::LoggingFlag::DisableApplicationAbortOnFatalLog)";
02650             base::utils::abort(1, reasonStream.str());
02651         }
02652         m_proceed = false;
02653     } catch (std::exception & ex) {
02654         // Extremely low memory situation; don't let exception be unhandled.
02655     }
02656 }
02657
02658 // PErrorWriter
02659
02660 PErrorWriter::~PErrorWriter(void) {
02661     if (m_proceed) {
02662 #if ELPP_COMPILER_MSVC
02663         char buff[256];
02664         strerror_s(buff, 256, errno);
02665         m_logger->stream() << ": " << buff << " [" << errno << "];"
02666     } else
02667     m_logger->stream() << ": " << strerror(errno) << " [" << errno << "];"
02668 #endif

```

```

02669     }
02670 }
02671
02672 // PerformanceTracker
02673
02674 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02675
02676 PerformanceTracker::PerformanceTracker(const std::string& blockName,
02677                                         base::TimestampUnit timestampUnit,
02678                                         const std::string& loggerId,
02679                                         bool scopedLog, Level level) :
02680     m_blockName(blockName), m_timestampUnit(timestampUnit), m_loggerId(loggerId),
02681     m_scopedLog(scopedLog),
02682     m_level(level), m_hasChecked(false), m_lastCheckpointId(std::string()), m_enabled(false) {
02683     #if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
02684         // We store it locally so that if user happen to change configuration by the end of scope
02685         // or before calling checkpoint, we still depend on state of configuration at time of construction
02686         el::Logger* loggerPtr = ELPP->registeredLoggers()->get(loggerId, false);
02687         m_enabled = loggerPtr != nullptr && loggerPtr->m_typedConfigurations->performanceTracking(m_level);
02688         if (m_enabled) {
02689             base::utils::DateTime::gettimeofday(&m_startTime);
02690         }
02691     #endif // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
02692 }
02693
02694 PerformanceTracker::~PerformanceTracker(void) {
02695     #if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
02696         if (m_enabled) {
02697             base::threading::ScopedLock scopedLock(lock());
02698             if (m_scopedLog) {
02699                 base::utils::DateTime::gettimeofday(&m_endTime);
02700                 base::type::string_t formattedTime = getFormattedTimeTaken();
02701                 PerformanceTrackingData data(PerformanceTrackingData::DataType::Complete);
02702                 data.init(this);
02703                 data.m_formattedTimeTaken = formattedTime;
02704                 PerformanceTrackingCallback* callback = nullptr;
02705                 for (const std::pair<std::string, base::type::PerformanceTrackingCallbackPtr>& h
02706                     : ELPP->m_performanceTrackingCallbacks) {
02707                     callback = h.second.get();
02708                     if (callback != nullptr && callback->enabled()) {
02709                         callback->handle(&data);
02710                     }
02711                 }
02712             }
02713         }
02714     #endif // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING)
02715 }
02716
02717 void PerformanceTracker::checkpoint(const std::string& id, const char* file, base::type::LineNumber
02718 line,
02719                                     const char* func) {
02720     #if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
02721         if (m_enabled) {
02722             base::threading::ScopedLock scopedLock(lock());
02723             base::utils::DateTime::gettimeofday(&m_endTime);
02724             base::type::string_t formattedTime = m_hasChecked ? getFormattedTimeTaken(m_lastCheckpointTime) :
02725             ELPP_LITERAL("");
02726             PerformanceTrackingData data(PerformanceTrackingData::DataType::Checkpoint);
02727             data.init(this);
02728             data.m_checkpointId = id;
02729             data.m_file = file;
02730             data.m_line = line;
02731             data.m_func = func;
02732             data.m_formattedTimeTaken = formattedTime;
02733             PerformanceTrackingCallback* callback = nullptr;
02734             for (const std::pair<std::string, base::type::PerformanceTrackingCallbackPtr>& h
02735                 : ELPP->m_performanceTrackingCallbacks) {
02736                 callback = h.second.get();
02737                 if (callback != nullptr && callback->enabled()) {
02738                     callback->handle(&data);
02739                 }
02740             }
02741             base::utils::DateTime::gettimeofday(&m_lastCheckpointTime);
02742             m_hasChecked = true;
02743             m_lastCheckpointId = id;
02744         }
02745     #endif // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
02746     ELPP_UNUSED(id);
02747     ELPP_UNUSED(file);
02748     ELPP_UNUSED(line);
02749     ELPP_UNUSED(func);
02750 }
02751
02752 const base::type::string_t PerformanceTracker::getFormattedTimeTaken(struct timeval startTime) const {
02753     if (ELPP->hasFlag(LoggingFlag::FixedTimeFormat)) {
02754         base::type::stringstream_t ss;
02755         ss << base::utils::DateTime::getTimeDifference(m_endTime,

```

```

02753         startTime, m_timestampUnit) << " " <<
base::consts::kTimeFormats[static_cast<base::type::EnumType>
02754         (m_timestampUnit)].unit;
02755     return ss.str();
02756 }
02757 return base::utils::DateTime::formatTime(base::utils::DateTime::getTimeDifference(m_endTime,
02758     startTime, m_timestampUnit), m_timestampUnit);
02759 }
02760
02761 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02762
02763 namespace debug {
02764 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
02765 // StackTrace
02766
02767 StackTrace::StackTraceEntry::StackTraceEntry(std::size_t index, const std::string& loc, const
std::string& demang,
02768     const std::string& hex,
02769     const std::string& addr) :
02770     m_index(index),
02771     m_location(loc),
02772     m_demangled(demang),
02773     m_hex(hex),
02774     m_addr(addr) {
02775 }
02776
02777 std::ostream& operator<<(std::ostream& ss, const StackTrace::StackTraceEntry& si) {
02778     ss << "[" << si.m_index << "]" << si.m_location << (si.m_hex.empty() ? "" : "+") << si.m_hex << " " <<
si.m_addr <<
02779     (si.m_demangled.empty() ? "" : ":") << si.m_demangled;
02780     return ss;
02781 }
02782
02783 std::ostream& operator<<(std::ostream& os, const StackTrace& st) {
02784     std::vector<StackTrace::StackTraceEntry>::const_iterator it = st.m_stack.begin();
02785     while (it != st.m_stack.end()) {
02786         os << "    " << *it++ << "\n";
02787     }
02788     return os;
02789 }
02790
02791 void StackTrace::generateNew(void) {
02792 #ifndef HAVE_EXECONF
02793     m_stack.clear();
02794     void* stack[kMaxStack];
02795     unsigned int size = backtrace(stack, kMaxStack);
02796     char** strings = backtrace_symbols(stack, size);
02797     if (size > kStackStart) { // Skip StackTrace c'tor and generateNew
02798         for (std::size_t i = kStackStart; i < size; ++i) {
02799             std::string mangName;
02800             std::string location;
02801             std::string hex;
02802             std::string addr;
02803
02804             // entry: 2    crash.cpp.bin    0x0000000101552be5
02805             _ZN2e14base5debug10StackTraceC1Ev + 21
02806             const std::string line(strings[i]);
02807             auto p = line.find("_");
02808             if (p != std::string::npos) {
02809                 mangName = line.substr(p);
02810                 mangName = mangName.substr(0, mangName.find(" +"));
02811             }
02812             p = line.find("0x");
02813             if (p != std::string::npos) {
02814                 addr = line.substr(p);
02815                 addr = addr.substr(0, addr.find("_"));
02816             }
02817             // Perform demangling if parsed properly
02818             if (!mangName.empty()) {
02819                 int status = 0;
02820                 char* demangName = abi::__cxa_demangle(mangName.data(), 0, 0, &status);
02821                 // if demangling is successful, output the demangled function name
02822                 if (status == 0) {
02823                     // Success (see
http://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.3/a01696.html)
02824                     StackTraceEntry entry(i - 1, location, demangName, hex, addr);
02825                     m_stack.push_back(entry);
02826                 } else {
02827                     // Not successful - we will use mangled name
02828                     StackTraceEntry entry(i - 1, location, mangName, hex, addr);
02829                     m_stack.push_back(entry);
02830                 }
02831                 free(demangName);
02832             } else {
02833                 StackTraceEntry entry(i - 1, line);
02834                 m_stack.push_back(entry);

```



```

02835     }
02836     }
02837 }
02838 free(strings);
02839 #else
02840     ELPP_INTERNAL_INFO(1, "Stacktrace generation not supported for selected compiler");
02841 #endif // ELPP_STACKTRACE
02842 }
02843
02844 // Static helper functions
02845
02846 static std::string crashReason(int sig) {
02847     std::stringstream ss;
02848     bool foundReason = false;
02849     for (int i = 0; i < base::consts::kCrashSignalsCount; ++i) {
02850         if (base::consts::kCrashSignals[i].numb == sig) {
02851             ss << "Application has crashed due to [" << base::consts::kCrashSignals[i].name << "] signal";
02852             if (ELPP->hasFlag(el::LoggingFlag::LogDetailedCrashReason)) {
02853                 ss << std::endl <<
02854                     "    " << base::consts::kCrashSignals[i].brief << std::endl <<
02855                     "    " << base::consts::kCrashSignals[i].detail;
02856             }
02857             foundReason = true;
02858         }
02859     }
02860     if (!foundReason) {
02861         ss << "Application has crashed due to unknown signal [" << sig << "]";
02862     }
02863     return ss.str();
02864 }
02865
02866 static void logCrashReason(int sig, bool stackTraceIfAvailable, Level level, const char* logger) {
02867     if (sig == SIGINT && ELPP->hasFlag(el::LoggingFlag::IgnoreSigInt)) {
02868         return;
02869     }
02870     std::stringstream ss;
02871     ss << "CRASH HANDLED; ";
02872     ss << crashReason(sig);
02873     #if ELPP_STACKTRACE
02874     if (stackTraceIfAvailable) {
02875         ss << std::endl << "    ===== Backtrace: =====" << std::endl << base::debug::StackTrace();
02876     }
02877     #else
02878     ELPP_UNUSED(stackTraceIfAvailable);
02879     #endif // ELPP_STACKTRACE
02880     ELPP_WRITE_LOG(el::base::Writer, level, base::DispatchAction::NormalLog, logger) << ss.str();
02881 }
02882
02883 static inline void crashAbort(int sig) {
02884     base::utils::abort(sig, std::string());
02885 }
02886
02887 static inline void defaultCrashHandler(int sig) {
02888     base::debug::logCrashReason(sig, true, Level::Fatal, base::consts::kDefaultLoggerId);
02889     base::debug::crashAbort(sig);
02890 }
02891
02892 // CrashHandler
02893
02894 CrashHandler::CrashHandler(bool useDefault) {
02895     if (useDefault) {
02896         setHandler(defaultCrashHandler);
02897     }
02898 }
02899
02900 void CrashHandler::setHandler(const Handler& cHandler) {
02901     m_handler = cHandler;
02902     #if defined(ELPP_HANDLE_SIGABRT)
02903     int i = 0; // SIGABRT is at base::consts::kCrashSignals[0]
02904     #else
02905     int i = 1;
02906     #endif // defined(ELPP_HANDLE_SIGABRT)
02907     for (; i < base::consts::kCrashSignalsCount; ++i) {
02908         m_handler = signal(base::consts::kCrashSignals[i].numb, cHandler);
02909     }
02910 }
02911
02912 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
02913 } // namespace debug
02914 } // namespace base
02915
02916 // el
02917
02918 // Helpers
02919
02920 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
02921 void Helpers::crashAbort(int sig, const char* sourceFile, unsigned int long line) {

```

```

02926     std::stringstream ss;
02927     ss << base::debug::crashReason(sig).c_str();
02928     ss << " - [Called el::Helpers::crashAbort(" < sig < ")]";
02929     if (sourceFile != nullptr && strlen(sourceFile) > 0) {
02930         ss << " - Source: " << sourceFile;
02931         if (line > 0)
02932             ss << ":" << line;
02933         else
02934             ss << " (line number not specified)";
02935     }
02936     base::utils::abort(sig, ss.str());
02937 }
02938
02939 void Helpers::logCrashReason(int sig, bool stackTraceIfAvailable, Level level, const char* logger) {
02940     el::base::debug::logCrashReason(sig, stackTraceIfAvailable, level, logger);
02941 }
02942
02943 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
02944
02945 // Loggers
02946
02947 Logger* Loggers::getLogger(const std::string& identity, bool registerIfNotAvailable) {
02948     return ELPP->registeredLoggers()->get(identity, registerIfNotAvailable);
02949 }
02950
02951 void Loggers::setDefaultLogBuilder(el::LogBuilderPtr& logBuilderPtr) {
02952     ELPP->registeredLoggers()->setDefaultLogBuilder(logBuilderPtr);
02953 }
02954
02955 bool Loggers::unregisterLogger(const std::string& identity) {
02956     return ELPP->registeredLoggers()->remove(identity);
02957 }
02958
02959 bool Loggers::hasLogger(const std::string& identity) {
02960     return ELPP->registeredLoggers()->has(identity);
02961 }
02962
02963 Logger* Loggers::reconfigureLogger(Logger* logger, const Configurations& configurations) {
02964     if (!logger) return nullptr;
02965     logger->configure(configurations);
02966     return logger;
02967 }
02968
02969 Logger* Loggers::reconfigureLogger(const std::string& identity, const Configurations& configurations) {
02970     return Loggers::reconfigureLogger(Loggers::getLogger(identity), configurations);
02971 }
02972
02973 Logger* Loggers::reconfigureLogger(const std::string& identity, ConfigurationType configurationType,
02974                                     const std::string& value) {
02975     Logger* logger = Loggers::getLogger(identity);
02976     if (logger == nullptr) {
02977         return nullptr;
02978     }
02979     logger->configurations()->set(Level::Global, configurationType, value);
02980     logger->reconfigure();
02981     return logger;
02982 }
02983
02984 void Loggers::reconfigureAllLoggers(const Configurations& configurations) {
02985     for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->begin();
02986          it != ELPP->registeredLoggers()->end(); ++it) {
02987         Loggers::reconfigureLogger(it->second, configurations);
02988     }
02989 }
02990
02991 void Loggers::reconfigureAllLoggers(Level level, ConfigurationType configurationType,
02992                                     const std::string& value) {
02993     for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->begin();
02994          it != ELPP->registeredLoggers()->end(); ++it) {
02995         Logger* logger = it->second;
02996         logger->configurations()->set(level, configurationType, value);
02997         logger->reconfigure();
02998     }
02999 }
03000
03001 void Loggers::setDefaultConfigurations(const Configurations& configurations, bool
reconfigureExistingLoggers) {
03002     ELPP->registeredLoggers()->setDefaultConfigurations(configurations);
03003     if (reconfigureExistingLoggers) {
03004         Loggers::reconfigureAllLoggers(configurations);
03005     }
03006 }
03007
03008 const Configurations* Loggers::defaultConfigurations(void) {
03009     return ELPP->registeredLoggers()->defaultConfigurations();
03010 }

```

```

03011
03012 const base::LogStreamsReferenceMapPtr Loggers::logStreamsReference(void) {
03013     return ELPP->registeredLoggers()->logStreamsReference();
03014 }
03015
03016 base::TypedConfigurations Loggers::defaultTypedConfigurations(void) {
03017     return base::TypedConfigurations(
03018         ELPP->registeredLoggers()->defaultConfigurations(),
03019         ELPP->registeredLoggers()->logStreamsReference());
03020 }
03021
03022 std::vector<std::string>* Loggers::populateAllLoggerIds(std::vector<std::string>* targetList) {
03023     targetList->clear();
03024     for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->list().begin();
03025         it != ELPP->registeredLoggers()->list().end(); ++it) {
03026         targetList->push_back(it->first);
03027     }
03028     return targetList;
03029 }
03030
03031 void Loggers::configureFromGlobal(const char* globalConfigurationFilePath) {
03032     std::ifstream gcfStream(globalConfigurationFilePath, std::ifstream::in);
03033     ELPP_ASSERT(gcfStream.is_open(), "Unable to open global configuration file [" <
03034         globalConfigurationFilePath
03035         < " ] for parsing.");
03036     std::string line = std::string();
03037     std::stringstream ss;
03038     Logger* logger = nullptr;
03039     auto configure = [&](void) {
03040         ELPP_INTERNAL_INFO(8, "Configuring logger: '" < logger->id() < "' with configurations \n" <
03041         ss.str()
03042         < " \n-----");
03043         Configurations c;
03044         c.parseFromText(ss.str());
03045         logger->configure(c);
03046     };
03047     while (gcfStream.good()) {
03048         std::getline(gcfStream, line);
03049         ELPP_INTERNAL_INFO(1, "Parsing line: " < line);
03050         base::utils::Str::trim(line);
03051         if (Configurations::Parser::isComment(line)) continue;
03052         Configurations::Parser::ignoreComments(&line);
03053         base::utils::Str::trim(line);
03054         if (line.size() > 2 && base::utils::Str::startsWith(line,
03055             std::string(base::consts::kConfigurationLoggerId))) {
03056             if (!ss.str().empty() && logger != nullptr) {
03057                 configure();
03058             }
03059             ss.str(std::string(""));
03060             line = line.substr(2);
03061             base::utils::Str::trim(line);
03062             if (line.size() > 1) {
03063                 ELPP_INTERNAL_INFO(1, "Getting logger: '" < line < "'");
03064                 logger = getLogger(line);
03065             }
03066             else {
03067                 ss < line < "\n";
03068             }
03069         }
03070         if (!ss.str().empty() && logger != nullptr) {
03071             configure();
03072         }
03073     }
03074
03075     bool Loggers::configureFromArg(const char* argKey) {
03076         #if defined(ELPP_DISABLE_CONFIGURATION_FROM_PROGRAM_ARGS)
03077         ELPP_UNUSED(argKey);
03078         #else
03079         if (!Helpers::commandLineArgs()->hasParamWithValue(argKey)) {
03080             return false;
03081         }
03082         configureFromGlobal(Helpers::commandLineArgs()->getParamValue(argKey));
03083     #endif // defined(ELPP_DISABLE_CONFIGURATION_FROM_PROGRAM_ARGS)
03084     return true;
03085 }
03086
03087 void Loggers::flushAll(void) {
03088     ELPP->registeredLoggers()->flushAll();
03089 }
03090
03091 void Loggers::setVerboseLevel(base::type::VerboseLevel level) {
03092     ELPP->vRegistry()->setLevel(level);
03093 }
03094
03095 base::type::VerboseLevel Loggers::verboseLevel(void) {
03096     return ELPP->vRegistry()->level();
03097 }

```

```

03095
03096 void Loggers::setVModules(const char* modules) {
03097     if (ELPP->vRegistry()->vModulesEnabled()) {
03098         ELPP->vRegistry()->setModules(modules);
03099     }
03100 }
03101
03102 void Loggers::clearVModules(void) {
03103     ELPP->vRegistry()->clearModules();
03104 }
03105
03106 // VersionInfo
03107
03108 const std::string VersionInfo::version(void) {
03109     return std::string("9.97.1");
03110 }
03112 const std::string VersionInfo::releaseDate(void) {
03113     return std::string("Thu Jul 20 2023 13:45:52 GMT+1000");
03114 }
03115
03116 } // 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 elptime_r localtime_r`
- `#define elptime_s localtime_s`
- `#define elptime 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, oc-↵casion, 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, oc-↵casion, el::Level::Fatal, dispatchAction, __VA_ARGS__)`
- `#define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, oc-↵casion, 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::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__)`
- `#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::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)`
- `#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 DPCHECK(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) DPCHECK(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::AllowVerboselfModuleNotSpecified](#) = 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, dispatchAction, __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, dispatchAction, __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, dispatchAction, __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, dispatchAction, __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, dispatchAction, __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, dispatchAction, __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 ) DPCHECK(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) {\n    m_logger->stream() « msg;\n    if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {\n        m_logger->stream() « " ";\n    }\n    return *this;\n}
```

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(e1_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(e1_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 {\nnamespace base {\n    el::base::type::StoragePointer elStorage;\n}\n    el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\n}
```

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

<i>ContainerType</i>	Type of container e.g, MyList from WX_DECLARE_LIST(int, MyList); in wxwidgets
<i>SizeMethod</i>	Method used to get size of container.
<i>ElementInstance</i>	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.](#)

```

00001 //
00002 // Bismillah ar-Rahmaan ar-Raheem
00003 //
00004 // Easylogging++ v9.97.1
00005 // Single-header only, cross-platform logging library for C++ applications
00006 //
00007 // Copyright (c) 2012-present @abumq (Majid Q.)
00008 //
00009 // This library is released under the MIT Licence.
00010 // https://github.com/amrayn/easyloggingpp/blob/master/LICENSE
00011 //
00012
00013 #ifndef EASYLOGGINGPP_H
00014 #define EASYLOGGINGPP_H
00015 // Compilers and C++0x/C++11 Evaluation
00016 #if __cplusplus >= 201103L
00017 # define ELPP_CXX11 1
00018 #endif // __cplusplus >= 201103L
00019 #if (defined(__GNUC__))
00020 # define ELPP_COMPILER_GCC 1
00021 #else
00022 # define ELPP_COMPILER_GCC 0
00023 #endif
00024 #if ELPP_COMPILER_GCC
00025 # define ELPP_GCC_VERSION (__GNUC__ * 10000 \
00026 + __GNUC_MINOR__ * 100 \
00027 + __GNUC_PATCHLEVEL__)
00028 # if defined(__GXX_EXPERIMENTAL_CXX0X__)
00029 # define ELPP_CXX0X 1
00030 # endif
00031 #endif
00032 // Visual C++
00033 #if defined(_MSC_VER)
00034 # define ELPP_COMPILER_MSVC 1
00035 #else
00036 # define ELPP_COMPILER_MSVC 0
00037 #endif
00038 #define ELPP_CRT_DBG_WARNINGS ELPP_COMPILER_MSVC
00039 #if ELPP_COMPILER_MSVC
00040 # if (_MSC_VER == 1600)
00041 # define ELPP_CXX0X 1
00042 # elif (_MSC_VER >= 1700)
00043 # define ELPP_CXX11 1
00044 # endif
00045 #endif
00046 // Clang++
00047 #if (defined(__clang__) && (__clang__ == 1))
00048 # define ELPP_COMPILER_CLANG 1
00049 #else
00050 # define ELPP_COMPILER_CLANG 0
00051 #endif
00052 #if ELPP_COMPILER_CLANG
00053 # if __has_include(<thread>)
00054 # include <cstdlib> // Make __GLIBCXX__ defined when using libstdc++
00055 # if !defined(__GLIBCXX__) || __GLIBCXX__ >= 20150426
00056 # define ELPP_CLANG_SUPPORTS_THREAD
00057 # endif // !defined(__GLIBCXX__) || __GLIBCXX__ >= 20150426
00058 # endif // __has_include(<thread>)
00059 #endif
00060 #if (defined(__MINGW32__) || defined(__MINGW64__))
00061 # define ELPP_MINGW 1
00062 #else
00063 # define ELPP_MINGW 0
00064 #endif
00065 #if (defined(__CYGWIN__) && (__CYGWIN__ == 1))
00066 # define ELPP_CYGWIN 1
00067 #else
00068 # define ELPP_CYGWIN 0
00069 #endif
00070 #if (defined(__INTEL_COMPILER))
00071 # define ELPP_COMPILER_INTEL 1
00072 #else
00073 # define ELPP_COMPILER_INTEL 0
00074 #endif
00075 // Operating System Evaluation
00076 // Windows
00077 #if (defined(_WIN32) || defined(_WIN64))
00078 # define ELPP_OS_WINDOWS 1
00079 #else
00080 # define ELPP_OS_WINDOWS 0
00081 #endif
00082 // Linux

```

```

00083 #if (defined(__linux) || defined(__linux__))
00084 # define ELPP_OS_LINUX 1
00085 #else
00086 # define ELPP_OS_LINUX 0
00087 #endif
00088 #if (defined(__APPLE__))
00089 # define ELPP_OS_MAC 1
00090 #else
00091 # define ELPP_OS_MAC 0
00092 #endif
00093 #if (defined(__FreeBSD__) || defined(__FreeBSD_kernel__))
00094 # define ELPP_OS_FREEBSD 1
00095 #else
00096 # define ELPP_OS_FREEBSD 0
00097 #endif
00098 #if (defined(__sun))
00099 # define ELPP_OS_SOLARIS 1
00100 #else
00101 # define ELPP_OS_SOLARIS 0
00102 #endif
00103 #if (defined(_AIX))
00104 # define ELPP_OS_AIX 1
00105 #else
00106 # define ELPP_OS_AIX 0
00107 #endif
00108 #if (defined(__NetBSD__))
00109 # define ELPP_OS_NETBSD 1
00110 #else
00111 # define ELPP_OS_NETBSD 0
00112 #endif
00113 #if defined(__EMSCRIPTEN__)
00114 # define ELPP_OS_EMSCRIPTEN 1
00115 #else
00116 # define ELPP_OS_EMSCRIPTEN 0
00117 #endif
00118 #if (defined(__QNX__) || defined(__QNXNTO__))
00119 # define ELPP_OS_QNX 1
00120 #else
00121 # define ELPP_OS_QNX 0
00122 #endif
00123 // Unix
00124 #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))
00125 # define ELPP_OS_UNIX 1
00126 #else
00127 # define ELPP_OS_UNIX 0
00128 #endif
00129 #if (defined(__ANDROID__))
00130 # define ELPP_OS_ANDROID 1
00131 #else
00132 # define ELPP_OS_ANDROID 0
00133 #endif
00134 // Evaluating Cygwin as *nix OS
00135 #if !ELPP_OS_UNIX && !ELPP_OS_WINDOWS && ELPP_CYGWIN
00136 # undef ELPP_OS_UNIX
00137 # undef ELPP_OS_LINUX
00138 # define ELPP_OS_UNIX 1
00139 # define ELPP_OS_LINUX 1
00140 #endif // !ELPP_OS_UNIX && !ELPP_OS_WINDOWS && ELPP_CYGWIN
00141 #if !defined(ELPP_INTERNAL_DEBUGGING_OUT_INFO)
00142 # define ELPP_INTERNAL_DEBUGGING_OUT_INFO std::cout
00143 #endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
00144 #if !defined(ELPP_INTERNAL_DEBUGGING_OUT_ERROR)
00145 # define ELPP_INTERNAL_DEBUGGING_OUT_ERROR std::cerr
00146 #endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
00147 #if !defined(ELPP_INTERNAL_DEBUGGING_ENDL)
00148 # define ELPP_INTERNAL_DEBUGGING_ENDL std::endl
00149 #endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
00150 #if !defined(ELPP_INTERNAL_DEBUGGING_MSG)
00151 # define ELPP_INTERNAL_DEBUGGING_MSG(msg) msg
00152 #endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
00153 // Internal Assertions and errors
00154 #if !defined(ELPP_DISABLE_ASSERT)
00155 # if (defined(ELPP_DEBUG_ASSERT_FAILURE))
00156 #   define ELPP_ASSERT(expr, msg) if (!(expr)) { \
00157     std::stringstream internalInfoStream; internalInfoStream << msg; \
00158     ELPP_INTERNAL_DEBUGGING_OUT_ERROR \
00159     << "EASYLOGGING++ ASSERTION FAILED (LINE: " << __LINE__ << ") [" #expr << "] WITH MESSAGE \"" \
00160     << ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) << "\"" << ELPP_INTERNAL_DEBUGGING_ENDL; \
00161     base::utils::abort(1, \
00162     "ELPP Assertion failure, please define ELPP_DEBUG_ASSERT_FAILURE"); }
00163 #   else
00164 #   define ELPP_ASSERT(expr, msg) if (!(expr)) { \
00165     std::stringstream internalInfoStream; internalInfoStream << msg; \
00166     ELPP_INTERNAL_DEBUGGING_OUT_ERROR \
00167     << "ASSERTION FAILURE FROM EASYLOGGING++ (LINE: " << __LINE__ << ") [" #expr << "] WITH MESSAGE \"" << ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str())

```

```

    « "\ " \
00168 « ELPP_INTERNAL_DEBUGGING_ENDL; }
00169 # endif // (defined(ELPP_DEBUG_ASSERT_FAILURE))
00170 #else
00171 # define ELPP_ASSERT(x, y)
00172 #endif // (!defined(ELPP_DISABLE_ASSERT))
00173 #if ELPP_COMPILER_MSVC
00174 # define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR \
00175 { char buff[256]; strerror_s(buff, 256, errno); \
00176 ELPP_INTERNAL_DEBUGGING_OUT_ERROR « ": " « buff « " [" « errno « "]; } (void)0
00177 #else
00178 # define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR \
00179 ELPP_INTERNAL_DEBUGGING_OUT_ERROR « ": " « strerror(errno) « " [" « errno « "]; (void)0
00180 #endif // ELPP_COMPILER_MSVC
00181 #if defined(ELPP_DEBUG_ERRORS)
00182 # if !defined(ELPP_INTERNAL_ERROR)
00183 # define ELPP_INTERNAL_ERROR(msg, pe) { \
00184 std::stringstream internalInfoStream; internalInfoStream « "<ERROR> " « msg; \
00185 ELPP_INTERNAL_DEBUGGING_OUT_ERROR \
00186 « "ERROR FROM EASYLOGGING++ (LINE: " « __LINE__ « ") " \
00187 « ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) « ELPP_INTERNAL_DEBUGGING_ENDL; \
00188 if (pe) { ELPP_INTERNAL_DEBUGGING_OUT_ERROR « " "; ELPP_INTERNAL_DEBUGGING_WRITE_PERROR; } } (void)0
00189 # endif
00190 #else
00191 # undef ELPP_INTERNAL_INFO
00192 # define ELPP_INTERNAL_ERROR(msg, pe)
00193 #endif // defined(ELPP_DEBUG_ERRORS)
00194 #if (defined(ELPP_DEBUG_INFO))
00195 # if !defined(ELPP_INTERNAL_INFO_LEVEL)
00196 # define ELPP_INTERNAL_INFO_LEVEL 9
00197 # endif // !(defined(ELPP_INTERNAL_INFO_LEVEL))
00198 # if !defined(ELPP_INTERNAL_INFO)
00199 # define ELPP_INTERNAL_INFO(lvl, msg) { if (lvl <= ELPP_INTERNAL_INFO_LEVEL) { \
00200 std::stringstream internalInfoStream; internalInfoStream « "<INFO> " « msg; \
00201 ELPP_INTERNAL_DEBUGGING_OUT_INFO « ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) \
00202 « ELPP_INTERNAL_DEBUGGING_ENDL; } }
00203 # endif
00204 #else
00205 # undef ELPP_INTERNAL_INFO
00206 # define ELPP_INTERNAL_INFO(lvl, msg)
00207 #endif // (defined(ELPP_DEBUG_INFO))
00208 #if (defined(ELPP_FEATURE_ALL)) || (defined(ELPP_FEATURE_CRASH_LOG))
00209 # if (ELPP_COMPILER_GCC && !ELPP_MINGW && !ELPP_CYGWIN && !ELPP_OS_ANDROID && !ELPP_OS_EMSCRIPTEN &&
!ELPP_OS_QNX)
00210 # define ELPP_STACKTRACE 1
00211 # else
00212 # if ELPP_COMPILER_MSVC
00213 # pragma message("Stack trace not available for this compiler")
00214 # else
00215 # warning "Stack trace not available for this compiler";
00216 # endif // ELPP_COMPILER_MSVC
00217 # define ELPP_STACKTRACE 0
00218 # endif // ELPP_COMPILER_GCC
00219 #else
00220 # define ELPP_STACKTRACE 0
00221 #endif // (defined(ELPP_FEATURE_ALL)) || (defined(ELPP_FEATURE_CRASH_LOG))
00222 // Miscellaneous macros
00223 #define ELPP_UNUSED(x) (void)x
00224 #if ELPP_OS_UNIX
00225 // Log file permissions for unix-based systems
00226 # define ELPP_LOG_PERMS S_IRUSR | S_IWUSR | S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP | S_IWOTH | S_IXOTH
00227 #endif // ELPP_OS_UNIX
00228 #if defined(ELPP_AS_DLL) && ELPP_COMPILER_MSVC
00229 # if defined(ELPP_EXPORT_SYMBOLS)
00230 # define ELPP_EXPORT __declspec(dllexport)
00231 # else
00232 # define ELPP_EXPORT __declspec(dllimport)
00233 # endif // defined(ELPP_EXPORT_SYMBOLS)
00234 #else
00235 # define ELPP_EXPORT
00236 #endif // defined(ELPP_AS_DLL) && ELPP_COMPILER_MSVC
00237 // Some special functions that are VC++ specific
00238 #undef STRTOK
00239 #undef STRERROR
00240 #undef STRCAT
00241 #undef STRCPY
00242 #if ELPP_CRT_DBG_WARNINGS
00243 # define STRTOK(a, b, c) strtok_s(a, b, c)
00244 # define STRERROR(a, b, c) strerror_s(a, b, c)
00245 # define STRCAT(a, b, len) strcat_s(a, len, b)
00246 # define STRCPY(a, b, len) strcpy_s(a, len, b)
00247 #else
00248 # define STRTOK(a, b, c) strtok(a, b)
00249 # define STRERROR(a, b, c) strerror(c)
00250 # define STRCAT(a, b, len) strcat(a, b)
00251 # define STRCPY(a, b, len) strcpy(a, b)
00252 #endif

```

```

00253 // Compiler specific support evaluations
00254 #if (ELPP_MINGW && !defined(ELPP_FORCE_USE_STD_THREAD))
00255 # define ELPP_USE_STD_THREADING 0
00256 #else
00257 # if ((ELPP_COMPILER_CLANG && defined(ELPP_CLANG_SUPPORTS_THREAD)) || \
00258      (ELPP_COMPILER_CLANG && defined(ELPP_CXX11)) || \
00259      defined(ELPP_FORCE_USE_STD_THREAD))
00260 #   define ELPP_USE_STD_THREADING 1
00261 # else
00262 #   define ELPP_USE_STD_THREADING 0
00263 # endif
00264 #endif
00265 #undef ELPP_FINAL
00266 #if ELPP_COMPILER_INTEL || (ELPP_GCC_VERSION < 40702)
00267 # define ELPP_FINAL
00268 #else
00269 # define ELPP_FINAL final
00270 #endif // ELPP_COMPILER_INTEL || (ELPP_GCC_VERSION < 40702)
00271 #if defined(ELPP_EXPERIMENTAL_ASYNC)
00272 # define ELPP_ASYNC_LOGGING 1
00273 #else
00274 # define ELPP_ASYNC_LOGGING 0
00275 #endif // defined(ELPP_EXPERIMENTAL_ASYNC)
00276 #if defined(ELPP_THREAD_SAFE) || ELPP_ASYNC_LOGGING
00277 # define ELPP_THREADING_ENABLED 1
00278 #else
00279 # define ELPP_THREADING_ENABLED 0
00280 #endif // defined(ELPP_THREAD_SAFE) || ELPP_ASYNC_LOGGING
00281 // Function macro ELPP_FUNC
00282 #undef ELPP_FUNC
00283 #if ELPP_COMPILER_MSVC // Visual C++
00284 # define ELPP_FUNC __FUNCSIG__
00285 #elif ELPP_COMPILER_GCC // GCC
00286 # define ELPP_FUNC __PRETTY_FUNCTION__
00287 #elif ELPP_COMPILER_INTEL // Intel C++
00288 # define ELPP_FUNC __PRETTY_FUNCTION__
00289 #elif ELPP_COMPILER_CLANG // Clang++
00290 # define ELPP_FUNC __PRETTY_FUNCTION__
00291 #else
00292 # if defined(__func__)
00293 #   define ELPP_FUNC __func__
00294 # else
00295 #   define ELPP_FUNC ""
00296 # endif // defined(__func__)
00297 #endif // defined(_MSC_VER)
00298 #undef ELPP_VARIADIC_TEMPLATES_SUPPORTED
00299 // Keep following line commented until features are fixed
00300 #define ELPP_VARIADIC_TEMPLATES_SUPPORTED \
00301 (ELPP_COMPILER_GCC || ELPP_COMPILER_CLANG || ELPP_COMPILER_INTEL || (ELPP_COMPILER_MSVC && _MSC_VER >=
1800))
00302 // Logging Enable/Disable macros
00303 #if defined(ELPP_DISABLE_LOGS)
00304 #define ELPP_LOGGING_ENABLED 0
00305 #else
00306 #define ELPP_LOGGING_ENABLED 1
00307 #endif
00308 #if (!defined(ELPP_DISABLE_DEBUG_LOGS) && (ELPP_LOGGING_ENABLED))
00309 # define ELPP_DEBUG_LOG 1
00310 #else
00311 # define ELPP_DEBUG_LOG 0
00312 #endif // (!defined(ELPP_DISABLE_DEBUG_LOGS) && (ELPP_LOGGING_ENABLED))
00313 #if (!defined(ELPP_DISABLE_INFO_LOGS) && (ELPP_LOGGING_ENABLED))
00314 # define ELPP_INFO_LOG 1
00315 #else
00316 # define ELPP_INFO_LOG 0
00317 #endif // (!defined(ELPP_DISABLE_INFO_LOGS) && (ELPP_LOGGING_ENABLED))
00318 #if (!defined(ELPP_DISABLE_WARNING_LOGS) && (ELPP_LOGGING_ENABLED))
00319 # define ELPP_WARNING_LOG 1
00320 #else
00321 # define ELPP_WARNING_LOG 0
00322 #endif // (!defined(ELPP_DISABLE_WARNING_LOGS) && (ELPP_LOGGING_ENABLED))
00323 #if (!defined(ELPP_DISABLE_ERROR_LOGS) && (ELPP_LOGGING_ENABLED))
00324 # define ELPP_ERROR_LOG 1
00325 #else
00326 # define ELPP_ERROR_LOG 0
00327 #endif // (!defined(ELPP_DISABLE_ERROR_LOGS) && (ELPP_LOGGING_ENABLED))
00328 #if (!defined(ELPP_DISABLE_FATAL_LOGS) && (ELPP_LOGGING_ENABLED))
00329 # define ELPP_FATAL_LOG 1
00330 #else
00331 # define ELPP_FATAL_LOG 0
00332 #endif // (!defined(ELPP_DISABLE_FATAL_LOGS) && (ELPP_LOGGING_ENABLED))
00333 #if (!defined(ELPP_DISABLE_TRACE_LOGS) && (ELPP_LOGGING_ENABLED))
00334 # define ELPP_TRACE_LOG 1
00335 #else
00336 # define ELPP_TRACE_LOG 0
00337 #endif // (!defined(ELPP_DISABLE_TRACE_LOGS) && (ELPP_LOGGING_ENABLED))
00338 #if (!defined(ELPP_DISABLE_VERBOSE_LOGS) && (ELPP_LOGGING_ENABLED))

```



```

00339 # define ELPP_VERBOSE_LOG 1
00340 #else
00341 # define ELPP_VERBOSE_LOG 0
00342 #endif // (!defined(ELPP_DISABLE_VERBOSE_LOGS) && (ELPP_LOGGING_ENABLED))
00343 #if !(ELPP_CXX0X || ELPP_CXX11)
00344 #   error "C++0x (or higher) support not detected! (Is '-std=c++11' missing?)"
00345 #endif // (!ELPP_CXX0X || ELPP_CXX11)
00346 // Headers
00347 #if defined(ELPP_SYSLOG)
00348 #   include <syslog.h>
00349 #endif // defined(ELPP_SYSLOG)
00350 #include <ctime>
00351 #include <cstring>
00352 #include <cstdlib>
00353 #include <cctype>
00354 #include <cwchar>
00355 #include <csignal>
00356 #include <cerrno>
00357 #include <cstdarg>
00358 #if defined(ELPP_UNICODE)
00359 #   include <locale>
00360 #   if ELPP_OS_WINDOWS
00361 #       include <codecvt>
00362 #   endif // ELPP_OS_WINDOWS
00363 #endif // defined(ELPP_UNICODE)
00364 #ifndef HAVE_EXECINFO
00365 #   include <cxxabi.h>
00366 #   include <execinfo.h>
00367 #endif // ENABLE_EXECINFO
00368 #if ELPP_OS_ANDROID
00369 #   include <sys/system_properties.h>
00370 #endif // ELPP_OS_ANDROID
00371 #if ELPP_OS_UNIX
00372 #   include <sys/stat.h>
00373 #   include <sys/time.h>
00374 #elif ELPP_OS_WINDOWS
00375 #   include <direct.h>
00376 #   include <windows.h>
00377 #   if defined(WIN32_LEAN_AND_MEAN)
00378 #       if defined(ELPP_WINSOCK2)
00379 #           include <winsock2.h>
00380 #       else
00381 #           include <winsock.h>
00382 #       endif // defined(ELPP_WINSOCK2)
00383 #   endif // defined(WIN32_LEAN_AND_MEAN)
00384 #endif // ELPP_OS_UNIX
00385 #include <string>
00386 #include <vector>
00387 #include <map>
00388 #include <unordered_map>
00389 #include <utility>
00390 #include <functional>
00391 #include <algorithm>
00392 #include <fstream>
00393 #include <iostream>
00394 #include <sstream>
00395 #include <memory>
00396 #include <type_traits>
00397 #if ELPP_THREADING_ENABLED
00398 #   if ELPP_USE_STD_THREADING
00399 #       include <mutex>
00400 #       include <thread>
00401 #   else
00402 #       if ELPP_OS_UNIX
00403 #           include <pthread.h>
00404 #       endif // ELPP_OS_UNIX
00405 #   endif // ELPP_USE_STD_THREADING
00406 #endif // ELPP_THREADING_ENABLED
00407 #if ELPP_ASYNC_LOGGING
00408 #   if defined(ELPP_NO_SLEEP_FOR)
00409 #       include <unistd.h>
00410 #   endif // defined(ELPP_NO_SLEEP_FOR)
00411 #   include <thread>
00412 #   include <queue>
00413 #   include <condition_variable>
00414 #endif // ELPP_ASYNC_LOGGING
00415 #if defined(ELPP_STL_LOGGING)
00416 // For logging STL based templates
00417 #   include <list>
00418 #   include <queue>
00419 #   include <deque>
00420 #   include <set>
00421 #   include <bitset>
00422 #   include <stack>
00423 #   if defined(ELPP_LOG_STD_ARRAY)
00424 #       include <array>
00425 #   endif // defined(ELPP_LOG_STD_ARRAY)

```

```

00426 # if defined(ELPP_LOG_UNORDERED_SET)
00427 #     include <unordered_set>
00428 # endif // defined(ELPP_UNORDERED_SET)
00429 #endif // defined(ELPP_STL_LOGGING)
00430 #if defined(ELPP_QT_LOGGING)
00431 // For logging Qt based classes & templates
00432 # include <QString>
00433 # include <QByteArray>
00434 # include <QVector>
00435 # include <QList>
00436 # include <QPair>
00437 # include <QMap>
00438 # include <QQueue>
00439 # include <QSet>
00440 # include <QLinkedList>
00441 # include <QHash>
00442 # include <QMultiHash>
00443 # include <QStack>
00444 #endif // defined(ELPP_QT_LOGGING)
00445 #if defined(ELPP_BOOST_LOGGING)
00446 // For logging boost based classes & templates
00447 # include <boost/container/vector.hpp>
00448 # include <boost/container/stable_vector.hpp>
00449 # include <boost/container/list.hpp>
00450 # include <boost/container/deque.hpp>
00451 # include <boost/container/map.hpp>
00452 # include <boost/container/flat_map.hpp>
00453 # include <boost/container/set.hpp>
00454 # include <boost/container/flat_set.hpp>
00455 #endif // defined(ELPP_BOOST_LOGGING)
00456 #if defined(ELPP_WXWIDGETS_LOGGING)
00457 // For logging wxWidgets based classes & templates
00458 # include <wx/vector.h>
00459 #endif // defined(ELPP_WXWIDGETS_LOGGING)
00460 #if defined(ELPP_UTC_DATETIME)
00461 # define elpptime_r gmtime_r
00462 # define elpptime_s gmtime_s
00463 # define elptime gmtime
00464 #else
00465 # define elpptime_r localtime_r
00466 # define elpptime_s localtime_s
00467 # define elptime localtime
00468 #endif // defined(ELPP_UTC_DATETIME)
00469 // Forward declarations
00470 namespace el {
00471 class Logger;
00472 class LogMessage;
00473 class PerformanceTrackingData;
00474 class Loggers;
00475 class Helpers;
00476 template <typename T> class Callback;
00477 class LogDispatchCallback;
00478 class PerformanceTrackingCallback;
00479 class LoggerRegistrationCallback;
00480 class LogDispatchData;
00481 namespace base {
00482 class Storage;
00483 class RegisteredLoggers;
00484 class PerformanceTracker;
00485 class MessageBuilder;
00486 class Writer;
00487 class PErrorWriter;
00488 class LogDispatcher;
00489 class DefaultLogBuilder;
00490 class DefaultLogDispatchCallback;
00491 #if ELPP_ASYNC_LOGGING
00492 class AsyncLogDispatchCallback;
00493 class AsyncDispatchWorker;
00494 #endif // ELPP_ASYNC_LOGGING
00495 class DefaultPerformanceTrackingCallback;
00496 } // namespace base
00497 } // namespace el
00498 namespace el {
00499 namespace base {
00500 namespace type {
00501 #undef ELPP_LITERAL
00502 #undef ELPP_STRLEN
00503 #undef ELPP_COUT
00504 #if defined(ELPP_UNICODE)
00505 # define ELPP_LITERAL(txt) L##txt
00506 # define ELPP_STRLEN wcslen
00507 # if defined(ELPP_CUSTOM_COUT)
00508 # define ELPP_COUT ELPP_CUSTOM_COUT
00509 # else
00510 # define ELPP_COUT std::wcout
00511 # endif // defined(ELPP_CUSTOM_COUT)
00512 # else
00513 # define ELPP_COUT std::cout
00514 # endif // defined(ELPP_CUSTOM_COUT)
00515 typedef wchar_t char_t;

```

```

00516 typedef std::wstring string_t;
00517 typedef std::wstringstream stringstream_t;
00518 typedef std::wfstream fstream_t;
00519 typedef std::wostream ostream_t;
00520 #else
00521 # define ELPP_LITERAL(txt) txt
00522 # define ELPP_STRLEN strlen
00523 # if defined ELPP_CUSTOM_COUT
00524 #   define ELPP_COUT ELPP_CUSTOM_COUT
00525 # else
00526 #   define ELPP_COUT std::cout
00527 # endif // defined ELPP_CUSTOM_COUT
00528 typedef char char_t;
00529 typedef std::string string_t;
00530 typedef std::stringstream stringstream_t;
00531 typedef std::fstream fstream_t;
00532 typedef std::ostream ostream_t;
00533 #endif // defined(ELPP_UNICODE)
00534 #if defined(ELPP_CUSTOM_COUT_LINE)
00535 # define ELPP_COUT_LINE(logLine) ELPP_CUSTOM_COUT_LINE(logLine)
00536 #else
00537 # define ELPP_COUT_LINE(logLine) logLine « std::flush
00538 #endif // defined(ELPP_CUSTOM_COUT_LINE)
00539 typedef unsigned int EnumType;
00540 typedef unsigned short VerboseLevel;
00541 typedef unsigned long int LineNumber;
00542 typedef std::shared_ptr<base::Storage> StoragePointer;
00543 typedef std::shared_ptr<LogDispatchCallback> LogDispatchCallbackPtr;
00544 typedef std::shared_ptr<PerformanceTrackingCallback> PerformanceTrackingCallbackPtr;
00545 typedef std::shared_ptr<LoggerRegistrationCallback> LoggerRegistrationCallbackPtr;
00546 typedef std::unique_ptr<el::base::PerformanceTracker> PerformanceTrackerPtr;
00547 } // namespace type
00551 class NoCopy {
00552 protected:
00553 NoCopy(void) {}
00554 private:
00555 NoCopy(const NoCopy&);
00556 NoCopy& operator=(const NoCopy&);
00557 };
00562 class StaticClass {
00563 private:
00564 StaticClass(void);
00565 StaticClass(const StaticClass&);
00566 StaticClass& operator=(const StaticClass&);
00567 };
00568 } // namespace base
00573 enum class Level : base::type::EnumType {
00575 Global = 1,
00577 Trace = 2,
00579 Debug = 4,
00581 Fatal = 8,
00583 Error = 16,
00585 Warning = 32,
00587 Verbose = 64,
00589 Info = 128,
00591 Unknown = 1010
00592 };
00593 } // namespace el
00594 namespace std {
00595 template<> struct hash<el::Level> {
00596 public:
00597 std::size_t operator()(const el::Level& l) const {
00598 return hash<el::base::type::EnumType> {}(static_cast<el::base::type::EnumType>(l));
00599 }
00600 };
00601 }
00602 namespace el {
00604 class LevelHelper : base::StaticClass {
00605 public:
00607 static const base::type::EnumType kMinValid = static_cast<base::type::EnumType>(Level::Trace);
00609 static const base::type::EnumType kMaxValid = static_cast<base::type::EnumType>(Level::Info);
00611 static base::type::EnumType castToInt(Level level) {
00612 return static_cast<base::type::EnumType>(level);
00613 }
00615 static Level castFromInt(base::type::EnumType l) {
00616 return static_cast<Level>(l);
00617 }
00620 static const char* convertToString(Level level);
00624 static Level convertFromString(const char* levelStr);
00629 static void foreachLevel(base::type::EnumType* startIndex, const std::function<bool(void)>& fn);
00630 };
00633 enum class ConfigurationType : base::type::EnumType {
00636 Enabled = 1,
00638 ToFile = 2,
00641 ToStandardOutput = 4,
00643 Format = 8,
00645 Filename = 16,

```

```

00647     SubsecondPrecision = 32,
00649     MillisecondsWidth = SubsecondPrecision,
00653     PerformanceTracking = 64,
00658     MaxLogFileSize = 128,
00660     LogFlushThreshold = 256,
00662     Unknown = 1010
00663 };
00665 class ConfigurationTypeHelper : base::StaticClass {
00666 public:
00668     static const base::type::EnumType kMinValid =
00669         static_cast<base::type::EnumType>(ConfigurationType::Enabled);
00670     static const base::type::EnumType kMaxValid =
00671         static_cast<base::type::EnumType>(ConfigurationType::MaxLogFileSize);
00672     static base::type::EnumType castToInt(ConfigurationType configurationType) {
00673         return static_cast<base::type::EnumType>(configurationType);
00674     }
00676     static ConfigurationType castFromInt(base::type::EnumType c) {
00677         return static_cast<ConfigurationType>(c);
00678     }
00681     static const char* convertToString(ConfigurationType configurationType);
00685     static ConfigurationType convertFromString(const char* configStr);
00691     static inline void forEachConfigType(base::type::EnumType* startIndex, const
std::function<bool(void)>& fn);
00692 };
00694 enum class LoggingFlag : base::type::EnumType {
00696     NewLineForContainer = 1,
00699     AllowVerboseIfModuleNotSpecified = 2,
00701     LogDetailedCrashReason = 4,
00703     DisableApplicationAbortOnFatalLog = 8,
00705     ImmediateFlush = 16,
00707     StrictLogFileSizeCheck = 32,
00709     ColoredTerminalOutput = 64,
00711     MultiLoggerSupport = 128,
00713     DisablePerformanceTrackingCheckpointComparison = 256,
00715     DisableVModules = 512,
00717     DisableVModulesExtensions = 1024,
00719     HierarchicalLogging = 2048,
00721     CreateLoggerAutomatically = 4096,
00723     AutoSpacing = 8192,
00725     FixedTimeFormat = 16384,
00726     // @brief Ignore SIGINT or crash
00727     IgnoreSigInt = 32768,
00728 };
00729 namespace base {
00731 namespace consts {
00732     static const char kFormatSpecifierCharValue = 'v';
00733     static const char kFormatSpecifierChar = '%';
00734     static const unsigned int kMaxLogPerCounter = 100000;
00735     static const unsigned int kMaxLogPerContainer = 100;
00736     static const unsigned int kDefaultSubsecondPrecision = 3;
00737
00738 #ifdef ELPP_DEFAULT_LOGGER
00739     static const char* kDefaultLoggerId = ELPP_DEFAULT_LOGGER;
00740 #else
00741     static const char* kDefaultLoggerId = "default";
00742 #endif
00743
00744 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
00745 #ifdef ELPP_DEFAULT_PERFORMANCE_LOGGER
00746     static const char* kPerformanceLoggerId = ELPP_DEFAULT_PERFORMANCE_LOGGER;
00747 #else
00748     static const char* kPerformanceLoggerId = "performance";
00749 #endif // ELPP_DEFAULT_PERFORMANCE_LOGGER
00750 #endif
00751
00752 #if defined(ELPP_SYSLOG)
00753     static const char* kSysLogLoggerId = "syslog";
00754 #endif // defined(ELPP_SYSLOG)
00755
00756 #if ELPP_OS_WINDOWS
00757     static const char* kFilePathSeparator = "\\";
00758 #else
00759     static const char* kFilePathSeparator = "/";
00760 #endif // ELPP_OS_WINDOWS
00761
00762     static const std::size_t kSourceFilenameMaxLength = 100;
00763     static const std::size_t kSourceLineMaxLength = 10;
00764     static const Level kPerformanceTrackerDefaultLevel = Level::Info;
00765     const struct {
00766         double value;
00767         const base::type::char_t* unit;
00768     } kTimeFormats[] = {
00769         { 1000.0f, ELPP_LITERAL("us") },
00770         { 1000.0f, ELPP_LITERAL("ms") },
00771         { 60.0f, ELPP_LITERAL("seconds") },
00772         { 60.0f, ELPP_LITERAL("minutes") },
00773         { 24.0f, ELPP_LITERAL("hours") },

```

```

00774 { 7.0f, ELPP_LITERAL("days") }
00775 };
00776 static const int kTimeFormatsCount = sizeof(kTimeFormats) /
    sizeof(kTimeFormats[0]);
00777 const struct {
00778     int numb;
00779     const char* name;
00780     const char* brief;
00781     const char* detail;
00782 } kCrashSignals[] = {
00783     // NOTE: Do not re-order, if you do please check CrashHandler(bool) constructor and
    CrashHandler::setHandler(..)
00784 {
00785     SIGABRT, "SIGABRT", "Abnormal termination",
00786     "Program was abnormally terminated."
00787 },
00788 {
00789     SIGFPE, "SIGFPE", "Erroneous arithmetic operation",
00790     "Arithmetic operation issue such as division by zero or operation resulting in overflow."
00791 },
00792 {
00793     SIGILL, "SIGILL", "Illegal instruction",
00794     "Generally due to a corruption in the code or to an attempt to execute data."
00795 },
00796 {
00797     SIGSEGV, "SIGSEGV", "Invalid access to memory",
00798     "Program is trying to read an invalid (unallocated, deleted or corrupted) or inaccessible memory."
00799 },
00800 {
00801     SIGINT, "SIGINT", "Interactive attention signal",
00802     "Interruption generated (generally) by user or operating system."
00803 },
00804 };
00805 static const int kCrashSignalsCount = sizeof(kCrashSignals) /
    sizeof(kCrashSignals[0]);
00806 } // namespace consts
00807 } // namespace base
00808 typedef std::function<void(const char*, std::size_t)> PreRollOutCallback;
00809 namespace base {
00810 static inline void defaultPreRollOutCallback(const char*, std::size_t) {}
00812 enum class TimestampUnit : base::type::EnumType {
00813     Microsecond = 0, Millisecond = 1, Second = 2, Minute = 3, Hour = 4, Day = 5
00814 };
00816 enum class FormatFlags : base::type::EnumType {
00817     DateTime = 1 << 1,
00818     LoggerId = 1 << 2,
00819     File = 1 << 3,
00820     Line = 1 << 4,
00821     Location = 1 << 5,
00822     Function = 1 << 6,
00823     User = 1 << 7,
00824     Host = 1 << 8,
00825     LogMessage = 1 << 9,
00826     VerboseLevel = 1 << 10,
00827     AppName = 1 << 11,
00828     ThreadId = 1 << 12,
00829     Level = 1 << 13,
00830     FileBase = 1 << 14,
00831     LevelShort = 1 << 15
00832 };
00834 class SubsecondPrecision {
00835 public:
00836     SubsecondPrecision(void) {
00837         init(base::consts::kDefaultSubsecondPrecision);
00838     }
00839     explicit SubsecondPrecision(int width) {
00840         init(width);
00841     }
00842     bool operator==(const SubsecondPrecision& ssPrec) {
00843         return m_width == ssPrec.m_width && m_offset == ssPrec.m_offset;
00844     }
00845     int m_width;
00846     unsigned int m_offset;
00847 private:
00848     void init(int width);
00849 };
00851 typedef SubsecondPrecision MillisecondsWidth;
00853 namespace utils {
00855 template <typename T>
00856 static
00857 typename std::enable_if<std::is_pointer<T*>::value, void>::type
00858 safeDelete(T*& pointer) {
00859     if (pointer == nullptr)
00860         return;
00861     delete pointer;
00862     pointer = nullptr;
00863 }

```

```

00866 namespace bitwise {
00867 template <typename Enum>
00868 static inline base::type::EnumType And(Enum e, base::type::EnumType flag) {
00869     return static_cast<base::type::EnumType>(flag) & static_cast<base::type::EnumType>(e);
00870 }
00871 template <typename Enum>
00872 static inline base::type::EnumType Not(Enum e, base::type::EnumType flag) {
00873     return static_cast<base::type::EnumType>(flag) & ~(static_cast<base::type::EnumType>(e));
00874 }
00875 template <typename Enum>
00876 static inline base::type::EnumType Or(Enum e, base::type::EnumType flag) {
00877     return static_cast<base::type::EnumType>(flag) | static_cast<base::type::EnumType>(e);
00878 }
00879 } // namespace bitwise
00880 template <typename Enum>
00881 static inline void addFlag(Enum e, base::type::EnumType* flag) {
00882     *flag = base::utils::bitwise::Or<Enum>(e, *flag);
00883 }
00884 template <typename Enum>
00885 static inline void removeFlag(Enum e, base::type::EnumType* flag) {
00886     *flag = base::utils::bitwise::Not<Enum>(e, *flag);
00887 }
00888 template <typename Enum>
00889 static inline bool hasFlag(Enum e, base::type::EnumType flag) {
00890     return base::utils::bitwise::And<Enum>(e, flag) > 0x0;
00891 }
00892 } // namespace utils
00893 namespace threading {
00894 #if ELPP_THREADING_ENABLED
00895 #   if !ELPP_USE_STD_THREADING
00896 namespace internal {
00897 class Mutex : base::NoCopy {
00898 public:
00900     Mutex(void) {
00901 #   if ELPP_OS_UNIX
00902         pthread_mutexattr_t attr;
00903         pthread_mutexattr_init(&attr);
00904         pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_RECURSIVE);
00905         pthread_mutex_init(&m_underlyingMutex, &attr);
00906         pthread_mutexattr_destroy(&attr);
00907 #   elif ELPP_OS_WINDOWS
00908         InitializeCriticalSection(&m_underlyingMutex);
00909 #   endif // ELPP_OS_UNIX
00910     }
00911     virtual ~Mutex(void) {
00912 #   if ELPP_OS_UNIX
00913         pthread_mutex_destroy(&m_underlyingMutex);
00914 #   elif ELPP_OS_WINDOWS
00915         DeleteCriticalSection(&m_underlyingMutex);
00916 #   endif // ELPP_OS_UNIX
00917     }
00918     inline void lock(void) {
00919 #   if ELPP_OS_UNIX
00920         pthread_mutex_lock(&m_underlyingMutex);
00921 #   elif ELPP_OS_WINDOWS
00922         EnterCriticalSection(&m_underlyingMutex);
00923 #   endif // ELPP_OS_UNIX
00924     }
00925     inline bool try_lock(void) {
00926 #   if ELPP_OS_UNIX
00927         return (pthread_mutex_trylock(&m_underlyingMutex) == 0);
00928 #   elif ELPP_OS_WINDOWS
00929         return TryEnterCriticalSection(&m_underlyingMutex);
00930 #   endif // ELPP_OS_UNIX
00931     }
00932     inline void unlock(void) {
00933 #   if ELPP_OS_UNIX
00934         pthread_mutex_unlock(&m_underlyingMutex);
00935 #   elif ELPP_OS_WINDOWS
00936         LeaveCriticalSection(&m_underlyingMutex);
00937 #   endif // ELPP_OS_UNIX
00938     }
00939 private:
00940 #   if ELPP_OS_UNIX
00941         pthread_mutex_t m_underlyingMutex;
00942 #   elif ELPP_OS_WINDOWS
00943         CRITICAL_SECTION m_underlyingMutex;
00944 #   endif // ELPP_OS_UNIX
00945 };
00946 template <typename M>
00947 class ScopedLock : base::NoCopy {
00948 public:

```

```

00955     explicit ScopedLock(M& mutex) {
00956         m_mutex = &mutex;
00957         m_mutex->lock();
00958     }
00959
00960     virtual ~ScopedLock(void) {
00961         m_mutex->unlock();
00962     }
00963 private:
00964     M* m_mutex;
00965     ScopedLock(void);
00966 };
00967 } // namespace internal
00968 typedef base::threading::internal::Mutex Mutex;
00969 typedef base::threading::internal::ScopedLock<base::threading::Mutex> ScopedLock;
00970 # else
00971 typedef std::recursive_mutex Mutex;
00972 typedef std::lock_guard<base::threading::Mutex> ScopedLock;
00973 # endif // !ELPP_USE_STD_THREADING
00974 #else
00975 namespace internal {
00976 class NoMutex : base::NoCopy {
00977 public:
00978     NoMutex(void) {}
00979     inline void lock(void) {}
00980     inline bool try_lock(void) {
00981         return true;
00982     }
00983     inline void unlock(void) {}
00984 };
00985
00986 template <typename Mutex>
00987 class NoScopedLock : base::NoCopy {
00988 public:
00989     explicit NoScopedLock(Mutex&) {
00990     }
00991     virtual ~NoScopedLock(void) {
00992     }
00993 private:
00994     NoScopedLock(void);
00995 };
00996 } // namespace internal
00997 typedef base::threading::internal::NoMutex Mutex;
00998 typedef base::threading::internal::NoScopedLock<base::threading::Mutex> ScopedLock;
01000 #endif // ELPP_THREADING_ENABLED
01002 class ThreadSafe {
01003 public:
01004     virtual inline void acquireLock(void) ELPP_FINAL { m_mutex.lock(); }
01005     virtual inline void releaseLock(void) ELPP_FINAL { m_mutex.unlock(); }
01006     virtual inline base::threading::Mutex& lock(void) ELPP_FINAL { return m_mutex; }
01007 protected:
01008     ThreadSafe(void) {}
01009     virtual ~ThreadSafe(void) {}
01010 private:
01011     base::threading::Mutex m_mutex;
01012 };
01013
01014 #if ELPP_THREADING_ENABLED
01015 # if !ELPP_USE_STD_THREADING
01017 static std::string getCurrentThreadId(void) {
01018     std::stringstream ss;
01019     # if (ELPP_OS_WINDOWS)
01020     ss << GetCurrentThreadId();
01021     # endif // (ELPP_OS_WINDOWS)
01022     return ss.str();
01023 }
01024 # else
01026 static std::string getCurrentThreadId(void) {
01027     std::stringstream ss;
01028     ss << std::this_thread::get_id();
01029     return ss.str();
01030 }
01031 # endif // !ELPP_USE_STD_THREADING
01032 #else
01033 static inline std::string getCurrentThreadId(void) {
01034     return std::string();
01035 }
01036 #endif // ELPP_THREADING_ENABLED
01037 } // namespace threading
01038 namespace utils {
01039 class File : base::StaticClass {
01040 public:
01041     static base::type::fstream_t* newFileStream(const std::string& filename);
01042
01043     static std::size_t getToFileSize(base::type::fstream_t* fs);
01044
01045     static bool pathExists(const char* path, bool considerFile = false);
01050

```

```

01053     static bool createPath(const std::string& path);
01055     static std::string extractPathFromFilename(const std::string& fullPath,
01056         const char* separator = base::consts::kFilePathSeparator);
01058     static void buildStrippedFilename(const char* filename, char buff[],
01059         std::size_t limit = base::consts::kSourceFilenameMaxLength);
01061     static void buildBaseFilename(const std::string& fullPath, char buff[],
01062         std::size_t limit = base::consts::kSourceFilenameMaxLength,
01063         const char* separator = base::consts::kFilePathSeparator);
01064 };
01066 class Str : base::StaticClass {
01067 public:
01069     static inline bool isDigit(char c) {
01070         return c >= '0' && c <= '9';
01071     }
01072
01074     static bool wildCardMatch(const char* str, const char* pattern);
01075
01076     static std::string& ltrim(std::string& str);
01077     static std::string& rtrim(std::string& str);
01078     static std::string& trim(std::string& str);
01079
01084     static bool startsWith(const std::string& str, const std::string& start);
01085
01090     static bool endsWith(const std::string& str, const std::string& end);
01091
01097     static std::string& replaceAll(std::string& str, char replaceWhat, char replaceWith);
01098
01104     static std::string& replaceAll(std::string& str, const std::string& replaceWhat,
01105         const std::string& replaceWith);
01106
01107     static void replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t&
01108         replaceWhat, const base::type::string_t& replaceWith);
01109 #if defined(ELPP_UNICODE)
01110     static void replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t&
01111         replaceWhat, const std::string& replaceWith);
01112 #endif // defined(ELPP_UNICODE)
01116     static std::string& toUpper(std::string& str);
01117
01119     static bool cStringEq(const char* s1, const char* s2);
01120
01123     static bool cStringCaseEq(const char* s1, const char* s2);
01124
01126     static bool contains(const char* str, char c);
01127
01128     static char* convertAndAddToBuff(std::size_t n, int len, char* buf, const char* bufLim, bool
01129         zeroPadded = true);
01129     static char* addToBuff(const char* str, char* buf, const char* bufLim);
01130     static char* clearBuff(char buff[], std::size_t lim);
01131
01134     static char* wcharPtrToCharPtr(const wchar_t* line);
01135 };
01137 class OS : base::StaticClass {
01138 public:
01139 #if ELPP_OS_WINDOWS
01144     static const char* getWindowsEnvironmentVariable(const char* varname);
01145 #endif // ELPP_OS_WINDOWS
01146 #if ELPP_OS_ANDROID
01148     static std::string getProperty(const char* prop);
01149
01151     static std::string getDeviceName(void);
01152 #endif // ELPP_OS_ANDROID
01153
01159     static const std::string getBashOutput(const char* command);
01160
01166     static std::string getEnvironmentVariable(const char* variableName, const char* defaultVal,
01167         const char* alternativeBashCommand = nullptr);
01169     static std::string currentUser(void);
01170
01174     static std::string currentHost(void);
01176     static bool termSupportsColor(void);
01177 };
01179 class DateTime : base::StaticClass {
01180 public:
01185     static void gettimeofday(struct timeval* tv);
01186
01191     static std::string getDateTime(const char* format, const base::SubsecondPrecision* ssPrec);
01192
01194     static std::string timevalToString(struct timeval tval, const char* format,
01195         const el::base::SubsecondPrecision* ssPrec);
01196
01198     static base::type::string_t formatTime(unsigned long long time, base::TimestampUnit timestampUnit);
01199
01201     static unsigned long long getTimeDifference(const struct timeval& endTime, const struct timeval&
01202         startTime, base::TimestampUnit timestampUnit);

```



```

01203
01204
01205     static struct ::tm* buildTimeInfo(struct timeval* currTime, struct ::tm* timeInfo);
01206 private:
01207     static char* parseFormat(char* buf, std::size_t bufSz, const char* format, const struct tm* tInfo,
01208                               std::size_t msec, const base::SubsecondPrecision* ssPrec);
01209 };
01211 class CommandLineArgs {
01212 public:
01213     CommandLineArgs(void) {
01214         setArgs(0, static_cast<char**>(nullptr));
01215     }
01216     CommandLineArgs(int argc, const char** argv) {
01217         setArgs(argc, argv);
01218     }
01219     CommandLineArgs(int argc, char** argv) {
01220         setArgs(argc, argv);
01221     }
01222     virtual ~CommandLineArgs(void) {}
01224     inline void setArgs(int argc, const char** argv) {
01225         setArgs(argc, const_cast<char**>(argv));
01226     }
01228     void setArgs(int argc, char** argv);
01230     bool hasParamWithValue(const char* paramKey) const;
01233     const char* getParamValue(const char* paramKey) const;
01235     bool hasParam(const char* paramKey) const;
01237     bool empty(void) const;
01239     std::size_t size(void) const;
01240     friend base::type::ostream_t& operator<<(base::type::ostream_t& os, const CommandLineArgs& c);
01241
01242 private:
01243     int m_argc;
01244     char** m_argv;
01245     std::unordered_map<std::string, std::string> m_paramsWithValue;
01246     std::vector<std::string> m_params;
01247 };
01254 template <typename T_Ptr, typename Container>
01255 class AbstractRegistry : public base::threading::ThreadSafe {
01256 public:
01257     typedef typename Container::iterator iterator;
01258     typedef typename Container::const_iterator const_iterator;
01259
01261     AbstractRegistry(void) {}
01262
01264     AbstractRegistry(AbstractRegistry&& sr) {
01265         if (this == &sr) {
01266             return;
01267         }
01268         unregisterAll();
01269         m_list = std::move(sr.m_list);
01270     }
01271
01272     bool operator==(const AbstractRegistry<T_Ptr, Container>& other) {
01273         if (size() != other.size()) {
01274             return false;
01275         }
01276         for (std::size_t i = 0; i < m_list.size(); ++i) {
01277             if (m_list.at(i) != other.m_list.at(i)) {
01278                 return false;
01279             }
01280         }
01281         return true;
01282     }
01283
01284     bool operator!=(const AbstractRegistry<T_Ptr, Container>& other) {
01285         if (size() != other.size()) {
01286             return true;
01287         }
01288         for (std::size_t i = 0; i < m_list.size(); ++i) {
01289             if (m_list.at(i) != other.m_list.at(i)) {
01290                 return true;
01291             }
01292         }
01293         return false;
01294     }
01295
01297     AbstractRegistry& operator=(AbstractRegistry&& sr) {
01298         if (this == &sr) {
01299             return *this;
01300         }
01301         unregisterAll();
01302         m_list = std::move(sr.m_list);
01303         return *this;
01304     }
01305
01306     virtual ~AbstractRegistry(void) {}
01307 }

```

```

01308
01310     virtual inline iterator begin(void) ELPP_FINAL {
01311         return m_list.begin();
01312     }
01313
01315     virtual inline iterator end(void) ELPP_FINAL {
01316         return m_list.end();
01317     }
01318
01319
01321     virtual inline const_iterator cbegin(void) const ELPP_FINAL {
01322         return m_list.cbegin();
01323     }
01324
01326     virtual inline const_iterator cend(void) const ELPP_FINAL {
01327         return m_list.cend();
01328     }
01329
01331     virtual inline bool empty(void) const ELPP_FINAL {
01332         return m_list.empty();
01333     }
01334
01336     virtual inline std::size_t size(void) const ELPP_FINAL {
01337         return m_list.size();
01338     }
01339
01341     virtual inline Container& list(void) ELPP_FINAL {
01342         return m_list;
01343     }
01344
01346     virtual inline const Container& list(void) const ELPP_FINAL {
01347         return m_list;
01348     }
01349
01351     virtual void unregisterAll(void) = 0;
01352
01353 protected:
01354     virtual void deepCopy(const AbstractRegistry<T_Ptr, Container>&) = 0;
01355     void reinitDeepCopy(const AbstractRegistry<T_Ptr, Container>& sr) {
01356         unregisterAll();
01357         deepCopy(sr);
01358     }
01359
01360 private:
01361     Container m_list;
01362 };
01363
01369 template <typename T_Ptr, typename T_Key = const char*>
01370 class Registry : public AbstractRegistry<T_Ptr, std::unordered_map<T_Key, T_Ptr*>> {
01371 public:
01372     typedef typename Registry<T_Ptr, T_Key>::iterator iterator;
01373     typedef typename Registry<T_Ptr, T_Key>::const_iterator const_iterator;
01374
01375     Registry(void) {}
01376
01378     Registry(const Registry& sr) : AbstractRegistry<T_Ptr, std::vector<T_Ptr*>>() {
01379         if (this == &sr) {
01380             return;
01381         }
01382         this->reinitDeepCopy(sr);
01383     }
01384
01388     Registry& operator=(const Registry& sr) {
01389         if (this == &sr) {
01390             return *this;
01391         }
01392         this->reinitDeepCopy(sr);
01393         return *this;
01394     }
01395
01396     virtual ~Registry(void) {
01397         unregisterAll();
01398     }
01399
01400 protected:
01401     virtual void unregisterAll(void) ELPP_FINAL {
01402         if (!this->empty()) {
01403             for (auto&& curr : this->list()) {
01404                 base::utils::safeDelete(curr.second);
01405             }
01406             this->list().clear();
01407         }
01408     }
01409
01411     virtual void registerNew(const T_Key& uniqKey, T_Ptr* ptr) ELPP_FINAL {
01412         unregister(uniqKey);
01413         this->list().insert(std::make_pair(uniqKey, ptr));

```

```

01414     }
01415
01417     void unregister(const T_Key& uniqKey) {
01418         T_Ptr* existing = get(uniqKey);
01419         if (existing != nullptr) {
01420             this->list().erase(uniqKey);
01421             base::utils::safeDelete(existing);
01422         }
01423     }
01424
01426     T_Ptr* get(const T_Key& uniqKey) {
01427         iterator it = this->list().find(uniqKey);
01428         return it == this->list().end()
01429             ? nullptr
01430             : it->second;
01431     }
01432
01433 private:
01434     virtual void deepCopy(const AbstractRegistry<T_Ptr, std::unordered_map<T_Key, T_Ptr*>& sr)
01435     ELPP_FINAL {
01436         for (const_iterator it = sr.cbegin(); it != sr.cend(); ++it) {
01437             registerNew(it->first, new T_Ptr(*it->second));
01438         }
01439     };
01440
01445     template <typename T_Ptr, typename Pred>
01446     class RegistryWithPred : public AbstractRegistry<T_Ptr, std::vector<T_Ptr*> {
01447     public:
01448         typedef typename RegistryWithPred<T_Ptr, Pred>::iterator iterator;
01449         typedef typename RegistryWithPred<T_Ptr, Pred>::const_iterator const_iterator;
01450
01451         RegistryWithPred(void) {
01452         }
01453
01454         virtual ~RegistryWithPred(void) {
01455             unregisterAll();
01456         }
01457
01459         RegistryWithPred(const RegistryWithPred& sr) : AbstractRegistry<T_Ptr, std::vector<T_Ptr*>() {
01460             if (this == &sr) {
01461                 return;
01462             }
01463             this->reinitDeepCopy(sr);
01464         }
01465
01469         RegistryWithPred& operator=(const RegistryWithPred& sr) {
01470             if (this == &sr) {
01471                 return *this;
01472             }
01473             this->reinitDeepCopy(sr);
01474             return *this;
01475         }
01476
01477         friend base::type::ostream_t& operator<<(base::type::ostream_t& os, const RegistryWithPred& sr) {
01478             for (const_iterator it = sr.list().begin(); it != sr.list().end(); ++it) {
01479                 os << ELPP_LITERAL("    ") << **it << ELPP_LITERAL("\n");
01480             }
01481             return os;
01482         }
01483
01484     protected:
01485         virtual void unregisterAll(void) ELPP_FINAL {
01486             if (!this->empty()) {
01487                 for (auto& curr : this->list()) {
01488                     base::utils::safeDelete(curr);
01489                 }
01490                 this->list().clear();
01491             }
01492         }
01493
01494         virtual void unregister(T_Ptr* ptr) ELPP_FINAL {
01495             if (ptr) {
01496                 iterator iter = this->begin();
01497                 for (; iter != this->end(); ++iter) {
01498                     if (ptr == *iter) {
01499                         break;
01500                     }
01501                 }
01502                 if (iter != this->end() && *iter != nullptr) {
01503                     this->list().erase(iter);
01504                     base::utils::safeDelete(*iter);
01505                 }
01506             }
01507         }
01508
01509         virtual inline void registerNew(T_Ptr* ptr) ELPP_FINAL {

```

```

01510     this->list().push_back(ptr);
01511 }
01512
01515 template <typename T, typename T2>
01516 T_Ptr* get(const T& arg1, const T2 arg2) {
01517     iterator iter = std::find_if(this->list().begin(), this->list().end(), Pred(arg1, arg2));
01518     if (iter != this->list().end() && *iter != nullptr) {
01519         return *iter;
01520     }
01521     return nullptr;
01522 }
01523
01524 private:
01525 virtual void deepCopy(const AbstractRegistry<T_Ptr, std::vector<T_Ptr*>& sr) {
01526     for (const_iterator it = sr.list().begin(); it != sr.list().end(); ++it) {
01527         registerNew(new T_Ptr(**it));
01528     }
01529 }
01530 };
01531 class Utils {
01532 public:
01533     template <typename T, typename TPtr>
01534     static bool installCallback(const std::string& id, std::unordered_map<std::string, TPtr*> mapT) {
01535         if (mapT->find(id) == mapT->end()) {
01536             mapT->insert(std::make_pair(id, TPtr(new T())));
01537             return true;
01538         }
01539         return false;
01540     }
01541
01542     template <typename T, typename TPtr>
01543     static void uninstallCallback(const std::string& id, std::unordered_map<std::string, TPtr*> mapT) {
01544         if (mapT->find(id) != mapT->end()) {
01545             mapT->erase(id);
01546         }
01547     }
01548
01549     template <typename T, typename TPtr>
01550     static T* callback(const std::string& id, std::unordered_map<std::string, TPtr*> mapT) {
01551         typename std::unordered_map<std::string, TPtr*>::iterator iter = mapT->find(id);
01552         if (iter != mapT->end()) {
01553             return static_cast<T*>(iter->second.get());
01554         }
01555         return nullptr;
01556     }
01557 };
01558 } // namespace utils
01559 } // namespace base
01560 class Loggable {
01561 public:
01562     virtual ~Loggable(void) {}
01563     virtual void log(el::base::type::ostream_t& os) const = 0;
01564 private:
01565     friend inline el::base::type::ostream_t& operator<<(el::base::type::ostream_t& os, const Loggable& loggable) {
01566         loggable.log(os);
01567         return os;
01568     }
01569 };
01570 namespace base {
01571 class LogFormat : public Loggable {
01572 public:
01573     LogFormat(void);
01574     LogFormat(Level level, const base::type::string_t& format);
01575     LogFormat(const LogFormat& logFormat);
01576     LogFormat(LogFormat&& logFormat);
01577     LogFormat& operator=(const LogFormat& logFormat);
01578     virtual ~LogFormat(void) {}
01579     bool operator==(const LogFormat& other);
01580
01581     void parseFromFormat(const base::type::string_t& userFormat);
01582
01583     inline Level level(void) const {
01584         return m_level;
01585     }
01586
01587     inline const base::type::string_t& userFormat(void) const {
01588         return m_userFormat;
01589     }
01590
01591     inline const base::type::string_t& format(void) const {
01592         return m_format;
01593     }
01594
01595     inline const std::string& dateTimeFormat(void) const {
01596         return m_dateTimeFormat;
01597     }
01598 }

```

```

01604
01605     inline base::type::EnumType flags(void) const {
01606         return m_flags;
01607     }
01608
01609     inline bool hasFlag(base::FormatFlags flag) const {
01610         return base::utils::hasFlag(flag, m_flags);
01611     }
01612
01613     virtual void log(el::base::type::ostream_t& os) const {
01614         os << m_format;
01615     }
01616
01617 protected:
01621     virtual void updateDateFormat(std::size_t index, base::type::string_t& currFormat) ELPP_FINAL;
01622
01624     virtual void updateFormatSpec(void) ELPP_FINAL;
01625
01626     inline void addFlag(base::FormatFlags flag) {
01627         base::utils::addFlag(flag, &m_flags);
01628     }
01629
01630 private:
01631     Level m_level;
01632     base::type::string_t m_userFormat;
01633     base::type::string_t m_format;
01634     std::string m_dateTimeFormat;
01635     base::type::EnumType m_flags;
01636     std::string m_currentUser;
01637     std::string m_currentHost;
01638     friend class el::Logger; // To resolve loggerId format specifier easily
01639 };
01640 } // namespace base
01642 typedef std::function<std::string(const LogMessage*)> FormatSpecifierValueResolver;
01646 class CustomFormatSpecifier {
01647 public:
01648     CustomFormatSpecifier(const char* formatSpecifier, const FormatSpecifierValueResolver& resolver) :
01649         m_formatSpecifier(formatSpecifier), m_resolver(resolver) {}
01650     inline const char* formatSpecifier(void) const {
01651         return m_formatSpecifier;
01652     }
01653     inline const FormatSpecifierValueResolver& resolver(void) const {
01654         return m_resolver;
01655     }
01656     inline bool operator==(const char* formatSpecifier) {
01657         return strcmp(m_formatSpecifier, formatSpecifier) == 0;
01658     }
01659
01660 private:
01661     const char* m_formatSpecifier;
01662     FormatSpecifierValueResolver m_resolver;
01663 };
01673 class Configuration : public Loggable {
01674 public:
01675     Configuration(const Configuration& c);
01676     Configuration& operator=(const Configuration& c);
01677
01678     virtual ~Configuration(void) {}
01679
01680     Configuration(Level level, ConfigurationType configurationType, const std::string& value);
01681
01685     inline Level level(void) const {
01686         return m_level;
01687     }
01688
01690     inline ConfigurationType configurationType(void) const {
01691         return m_configurationType;
01692     }
01693
01695     inline const std::string& value(void) const {
01696         return m_value;
01697     }
01698
01702     inline void setValue(const std::string& value) {
01703         m_value = value;
01704     }
01705
01706     virtual void log(el::base::type::ostream_t& os) const;
01707
01709     class Predicate {
01710     public:
01711         Predicate(Level level, ConfigurationType configurationType);
01712
01713         bool operator()(const Configuration* conf) const;
01714
01715     private:

```

```

01716     Level m_level;
01717     ConfigurationType m_configurationType;
01718 };
01719
01720 private:
01721     Level m_level;
01722     ConfigurationType m_configurationType;
01723     std::string m_value;
01724 };
01725
01729 class Configurations : public base::utils::RegistryWithPred<Configuration, Configuration::Predicate> {
01730 public:
01731     Configurations(void);
01732
01733     Configurations(const std::string& configurationFile, bool useDefaultsForRemaining = true,
01734                   Configurations* base = nullptr);
01735
01743     virtual ~Configurations(void) {
01744     }
01745
01752     bool parseFromFile(const std::string& configurationFile, Configurations* base = nullptr);
01753
01762     bool parseFromText(const std::string& configurationsString, Configurations* base = nullptr);
01763
01766     void setFromBase(Configurations* base);
01767
01772     bool hasConfiguration(ConfigurationType configurationType);
01773
01777     bool hasConfiguration(Level level, ConfigurationType configurationType);
01778
01791     void set(Level level, ConfigurationType configurationType, const std::string& value);
01792
01795     void set(Configuration* conf);
01796
01797     inline Configuration* get(Level level, ConfigurationType configurationType) {
01798         base::threading::ScopedLock scopedLock(lock());
01799         return RegistryWithPred<Configuration, Configuration::Predicate>::get(level, configurationType);
01800     }
01801
01806     inline void setGlobally(ConfigurationType configurationType, const std::string& value) {
01807         setGlobally(configurationType, value, false);
01808     }
01809
01811     inline void clear(void) {
01812         base::threading::ScopedLock scopedLock(lock());
01813         unregisterAll();
01814     }
01815
01819     inline const std::string& configurationFile(void) const {
01820         return m_configurationFile;
01821     }
01822
01824     void setToDefault(void);
01825
01833     void setRemainingToDefault(void);
01834
01839     class Parser : base::StaticClass {
01840     public:
01841         static bool parseFromFile(const std::string& configurationFile, Configurations* sender,
01842                                 Configurations* base = nullptr);
01843
01844         static bool parseFromText(const std::string& configurationsString, Configurations* sender,
01845                                 Configurations* base = nullptr);
01846
01864     private:
01865         friend class el::Loggers;
01866         static void ignoreComments(std::string* line);
01867         static bool isLevel(const std::string& line);
01868         static bool isComment(const std::string& line);
01869         static inline bool isConfig(const std::string& line);
01870         static bool parseLine(std::string* line, std::string* currConfigStr, std::string* currLevelStr,
01871                               Level* currLevel,
01872                               Configurations* conf);
01873     };
01874
01875     private:
01876         std::string m_configurationFile;
01877         bool m_isFromFile;
01878         friend class el::Loggers;
01879
01880         void unsafeSetIfNotExist(Level level, ConfigurationType configurationType, const std::string&
01881                                value);
01882
01883         void unsafeSet(Level level, ConfigurationType configurationType, const std::string& value);
01884
01887         void setGlobally(ConfigurationType configurationType, const std::string& value, bool
01888                           includeGlobalLevel);

```

```

01888
01891     void unsafeSetGlobally(ConfigurationType configurationType, const std::string& value, bool
        includeGlobalLevel);
01892 };
01893
01894 namespace base {
01895 typedef std::shared_ptr<base::type::fstream_t> FileStreamPtr;
01896 typedef std::unordered_map<std::string, FileStreamPtr> LogStreamsReferenceMap;
01897 typedef std::shared_ptr<base::LogStreamsReferenceMap> LogStreamsReferenceMapPtr;
01904 class TypedConfigurations : public base::threading::ThreadSafe {
01905 public:
01909     TypedConfigurations(Configurations* configurations, LogStreamsReferenceMapPtr logStreamsReference);
01910
01911     TypedConfigurations(const TypedConfigurations& other);
01912
01913     virtual ~TypedConfigurations(void) {
01914     }
01915
01916     const Configurations* configurations(void) const {
01917         return m_configurations;
01918     }
01919
01920     bool enabled(Level level);
01921     bool toFile(Level level);
01922     const std::string& filename(Level level);
01923     bool toStandardOutput(Level level);
01924     const base::LogFormat& logFormat(Level level);
01925     const base::SubsecondPrecision& subsecondPrecision(Level level = Level::Global);
01926     const base::MillisecondsWidth& millisecondsWidth(Level level = Level::Global);
01927     bool performanceTracking(Level level = Level::Global);
01928     base::type::fstream_t* fileStream(Level level);
01929     std::size_t maxLogFileSize(Level level);
01930     std::size_t logFlushThreshold(Level level);
01931
01932 private:
01933     Configurations* m_configurations;
01934     std::unordered_map<Level, bool> m_enabledMap;
01935     std::unordered_map<Level, bool> m_toFileMap;
01936     std::unordered_map<Level, std::string> m_filenameMap;
01937     std::unordered_map<Level, bool> m_toStandardOutputMap;
01938     std::unordered_map<Level, base::LogFormat> m_logFormatMap;
01939     std::unordered_map<Level, base::SubsecondPrecision> m_subsecondPrecisionMap;
01940     std::unordered_map<Level, bool> m_performanceTrackingMap;
01941     std::unordered_map<Level, base::FileStreamPtr> m_fileStreamMap;
01942     std::unordered_map<Level, std::size_t> m_maxLogFileSizeMap;
01943     std::unordered_map<Level, std::size_t> m_logFlushThresholdMap;
01944     LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;
01945
01946     friend class el::Helpers;
01947     friend class el::base::MessageBuilder;
01948     friend class el::base::Writer;
01949     friend class el::base::DefaultLogDispatchCallback;
01950     friend class el::base::LogDispatcher;
01951
01952     template <typename Conf_T>
01953     inline Conf_T getConfigByVal(Level level, const std::unordered_map<Level, Conf_T>* confMap, const
        char* confName) {
01954         base::threading::ScopedLock scopedLock(lock());
01955         return unsafeGetConfigByVal(level, confMap, confName); // This is not unsafe anymore - mutex
        locked in scope
01956     }
01957
01958     template <typename Conf_T>
01959     inline Conf_T& getConfigByRef(Level level, std::unordered_map<Level, Conf_T>* confMap, const char*
        confName) {
01960         base::threading::ScopedLock scopedLock(lock());
01961         return unsafeGetConfigByRef(level, confMap, confName); // This is not unsafe anymore - mutex
        locked in scope
01962     }
01963
01964     template <typename Conf_T>
01965     Conf_T unsafeGetConfigByVal(Level level, const std::unordered_map<Level, Conf_T>* confMap, const
        char* confName) {
01966         ELPP_UNUSED(confName);
01967         typename std::unordered_map<Level, Conf_T>::const_iterator it = confMap->find(level);
01968         if (it == confMap->end()) {
01969             try {
01970                 return confMap->at(Level::Global);
01971             } catch (...) {
01972                 ELPP_INTERNAL_ERROR("Unable to get configuration [" « confName « "] for level ["
                    « LevelHelper::convertToString(level) « "]"
                    « std::endl « "Please ensure you have properly configured logger.",
01973                     false);
01974             }
01975             return Conf_T();
01976         }
01977     }
01978     return it->second;

```

```

01979     }
01980
01981     template <typename Conf_T>
01982     Conf_T& unsafeGetConfigByRef(Level level, std::unordered_map<Level, Conf_T>* confMap, const char*
01983     confName) {
01984         ELPP_UNUSED(confName);
01985         typename std::unordered_map<Level, Conf_T>::iterator it = confMap->find(level);
01986         if (it == confMap->end()) {
01987             try {
01988                 return confMap->at(Level::Global);
01989             } catch (...) {
01990                 ELPP_INTERNAL_ERROR("Unable to get configuration [" « confName « "] for level ["
01991                 « LevelHelper::convertToString(level) « "]"
01992                 « std::endl « "Please ensure you have properly configured logger.",
01993                 false);
01994             }
01995             return it->second;
01996         }
01997     }
01998     template <typename Conf_T>
01999     void setValue(Level level, const Conf_T& value, std::unordered_map<Level, Conf_T>* confMap,
02000     bool includeGlobalLevel = true) {
02001         // If map is empty and we are allowed to add into generic level (Level::Global), do it!
02002         if (confMap->empty() && includeGlobalLevel) {
02003             confMap->insert(std::make_pair(Level::Global, value));
02004             return;
02005         }
02006         // If same value exist in generic level already, dont add it to explicit level
02007         typename std::unordered_map<Level, Conf_T>::iterator it = confMap->find(Level::Global);
02008         if (it != confMap->end() && it->second == value) {
02009             return;
02010         }
02011         // Now make sure we dont double up values if we really need to add it to explicit level
02012         it = confMap->find(level);
02013         if (it == confMap->end()) {
02014             // Value not found for level, add new
02015             confMap->insert(std::make_pair(level, value));
02016         } else {
02017             // Value found, just update value
02018             confMap->at(level) = value;
02019         }
02020     }
02021     void build(Configurations* configurations);
02022     unsigned long getULong(std::string confVal);
02023     std::string resolveFilename(const std::string& filename);
02024     void insertFile(Level level, const std::string& fullFilename);
02025     bool unsafeValidateFileRolling(Level level, const PreRollOutCallback& preRollOutCallback);
02026
02027     inline bool validateFileRolling(Level level, const PreRollOutCallback& preRollOutCallback) {
02028         base::threading::ScopedLock scopedLock(lock());
02029         return unsafeValidateFileRolling(level, preRollOutCallback);
02030     }
02031 };
02032
02033 class HitCounter {
02034 public:
02035     HitCounter(void) :
02036         m_filename(""),
02037         m_lineNumber(0),
02038         m_hitCounts(0) {
02039     }
02040
02041     HitCounter(const char* filename, base::type::LineNumber lineNumber) :
02042         m_filename(filename),
02043         m_lineNumber(lineNumber),
02044         m_hitCounts(0) {
02045     }
02046
02047     HitCounter(const HitCounter& hitCounter) :
02048         m_filename(hitCounter.m_filename),
02049         m_lineNumber(hitCounter.m_lineNumber),
02050         m_hitCounts(hitCounter.m_hitCounts) {
02051     }
02052
02053     HitCounter& operator=(const HitCounter& hitCounter) {
02054         if (&hitCounter != this) {
02055             m_filename = hitCounter.m_filename;
02056             m_lineNumber = hitCounter.m_lineNumber;
02057             m_hitCounts = hitCounter.m_hitCounts;
02058         }
02059         return *this;
02060     }
02061
02062     virtual ~HitCounter(void) {
02063     }
02064

```



```

02066 inline void resetLocation(const char* filename, base::type::LineNumber lineNumber) {
02067     m_filename = filename;
02068     m_lineNumber = lineNumber;
02069 }
02070
02072 inline void validateHitCounts(std::size_t n) {
02073     if (m_hitCounts >= base::consts::kMaxLogPerCounter) {
02074         m_hitCounts = (n >= 1 ? base::consts::kMaxLogPerCounter % n : 0);
02075     }
02076     ++m_hitCounts;
02077 }
02078
02079 inline const char* filename(void) const {
02080     return m_filename;
02081 }
02082
02083 inline base::type::LineNumber lineNumber(void) const {
02084     return m_lineNumber;
02085 }
02086
02087 inline std::size_t hitCounts(void) const {
02088     return m_hitCounts;
02089 }
02090
02091 inline void increment(void) {
02092     ++m_hitCounts;
02093 }
02094
02095 class Predicate {
02096 public:
02097     Predicate(const char* filename, base::type::LineNumber lineNumber)
02098         : m_filename(filename),
02099           m_lineNumber(lineNumber) {}
02100
02101     inline bool operator()(const HitCounter* counter) {
02102         return ((counter != nullptr) &&
02103             (strcmp(counter->m_filename, m_filename) == 0) &&
02104             (counter->m_lineNumber == m_lineNumber));
02105     }
02106
02107 private:
02108     const char* m_filename;
02109     base::type::LineNumber m_lineNumber;
02110 };
02111
02112 private:
02113     const char* m_filename;
02114     base::type::LineNumber m_lineNumber;
02115     std::size_t m_hitCounts;
02116 };
02118 class RegisteredHitCounters : public base::utils::RegistryWithPred<base::HitCounter,
base::HitCounter::Predicate> {
02119 public:
02122     bool validateEveryN(const char* filename, base::type::LineNumber lineNumber, std::size_t n);
02123
02126     bool validateAfterN(const char* filename, base::type::LineNumber lineNumber, std::size_t n);
02127
02130     bool validateNTimes(const char* filename, base::type::LineNumber lineNumber, std::size_t n);
02131
02133     inline const base::HitCounter* getCounter(const char* filename, base::type::LineNumber lineNumber) {
02134         base::threading::ScopedLock scopedLock(lock());
02135         return get(filename, lineNumber);
02136     }
02137 };
02139 enum class DispatchAction : base::type::EnumType {
02140     None = 1, NormalLog = 2, SysLog = 4
02141 };
02142 // namespace base
02143 template <typename T>
02144 class Callback : protected base::threading::ThreadSafe {
02145 public:
02146     Callback(void) : m_enabled(true) {}
02147     inline bool enabled(void) const {
02148         return m_enabled;
02149     }
02150     inline void setEnabled(bool enabled) {
02151         base::threading::ScopedLock scopedLock(lock());
02152         m_enabled = enabled;
02153     }
02154 protected:
02155     virtual void handle(const T* handlePtr) = 0;
02156 private:
02157     bool m_enabled;
02158 };
02159 class LogDispatchData {
02160 public:
02161     LogDispatchData() : m_logMessage(nullptr), m_dispatchAction(base::DispatchAction::None) {}

```

```

02162 inline const LogMessage* logMessage(void) const {
02163     return m_logMessage;
02164 }
02165 inline base::DispatchAction dispatchAction(void) const {
02166     return m_dispatchAction;
02167 }
02168 inline void setLogMessage(LogMessage* logMessage) {
02169     m_logMessage = logMessage;
02170 }
02171 inline void setDispatchAction(base::DispatchAction dispatchAction) {
02172     m_dispatchAction = dispatchAction;
02173 }
02174 private:
02175     LogMessage* m_logMessage;
02176     base::DispatchAction m_dispatchAction;
02177     friend class base::LogDispatcher;
02178 };
02179
02180 class LogDispatchCallback : public Callback<LogDispatchData> {
02181 protected:
02182     virtual void handle(const LogDispatchData* data);
02183     base::threading::Mutex& fileHandle(const LogDispatchData* data);
02184 private:
02185     friend class base::LogDispatcher;
02186     std::unordered_map<std::string, std::unique_ptr<base::threading::Mutex>> m_fileLocks;
02187     base::threading::Mutex m_fileLocksMapLock;
02188 };
02189 class PerformanceTrackingCallback : public Callback<PerformanceTrackingData> {
02190 private:
02191     friend class base::PerformanceTracker;
02192 };
02193 class LoggerRegistrationCallback : public Callback<Logger> {
02194 private:
02195     friend class base::RegisteredLoggers;
02196 };
02197 class LogBuilder : base::NoCopy {
02198 public:
02199     LogBuilder() : m_termSupportsColor(base::utils::OS::termSupportsColor()) {}
02200     virtual ~LogBuilder(void) {
02201         ELPP_INTERNAL_INFO(3, "Destroying log builder...")
02202     }
02203     virtual base::type::string_t build(const LogMessage* logMessage, bool appendNewLine) const = 0;
02204     void convertToColoredOutput(base::type::string_t* logLine, Level level);
02205 private:
02206     bool m_termSupportsColor;
02207     friend class el::base::DefaultLogDispatchCallback;
02208 };
02209 typedef std::shared_ptr<LogBuilder> LogBuilderPtr;
02213 class Logger : public base::threading::ThreadSafe, public Loggable {
02214 public:
02215     Logger(const std::string& id, base::LogStreamsReferenceMapPtr logStreamsReference);
02216     Logger(const std::string& id, const Configurations& configurations, base::LogStreamsReferenceMapPtr
logStreamsReference);
02217     Logger(const Logger& logger);
02218     Logger& operator=(const Logger& logger);
02219
02220     virtual ~Logger(void) {
02221         base::utils::safeDelete(m_typedConfigurations);
02222     }
02223
02224     virtual inline void log(el::base::type::ostream_t& os) const {
02225         os << m_id.c_str();
02226     }
02227
02228     void configure(const Configurations& configurations);
02229
02230     void reconfigure(void);
02231
02232     inline const std::string& id(void) const {
02233         return m_id;
02234     }
02235
02236     inline const std::string& parentApplicationName(void) const {
02237         return m_parentApplicationName;
02238     }
02239
02240     inline void setParentApplicationName(const std::string& parentApplicationName) {
02241         m_parentApplicationName = parentApplicationName;
02242     }
02243
02244     inline Configurations* configurations(void) {
02245         return &m_configurations;
02246     }
02247
02248     inline base::TypedConfigurations* typedConfigurations(void) {
02249         return m_typedConfigurations;
02250     }
02251
02252 }

```

```

02253
02254     static bool isValidId(const std::string& id);
02255
02256     void flush(void);
02257
02258     void flush(Level level, base::type::fstream_t* fs);
02259
02260     inline bool isFlushNeeded(Level level) {
02261         return ++m_unflushedCount.find(level)->second >= m_typedConfigurations->logFlushThreshold(level);
02262     }
02263
02264     inline LogBuilder* logBuilder(void) const {
02265         return m_logBuilder.get();
02266     }
02267
02268     inline void setLogBuilder(const LogBuilderPtr& logBuilder) {
02269         m_logBuilder = logBuilder;
02270     }
02271
02272     inline bool enabled(Level level) const {
02273         return m_typedConfigurations->enabled(level);
02274     }
02275
02276 #if ELPP_VARIADIC_TEMPLATES_SUPPORTED
02277 # define LOGGER_LEVEL_WRITERS_SIGNATURES(FUNCTION_NAME)\
02278 template <typename T, typename... Args>\
02279 inline void FUNCTION_NAME(const char*, const T&, const Args&...);\
02280 template <typename T>\
02281 inline void FUNCTION_NAME(const T&);
02282
02283 template <typename T, typename... Args>
02284 inline void verbose(int, const char*, const T&, const Args&...);
02285
02286 template <typename T>
02287 inline void verbose(int, const T&);
02288
02289 LOGGER_LEVEL_WRITERS_SIGNATURES(info)
02290 LOGGER_LEVEL_WRITERS_SIGNATURES(debug)
02291 LOGGER_LEVEL_WRITERS_SIGNATURES(warn)
02292 LOGGER_LEVEL_WRITERS_SIGNATURES(error)
02293 LOGGER_LEVEL_WRITERS_SIGNATURES(fatal)
02294 LOGGER_LEVEL_WRITERS_SIGNATURES(trace)
02295 # undef LOGGER_LEVEL_WRITERS_SIGNATURES
02296 #endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
02297 private:
02298     std::string m_id;
02299     base::TypedConfigurations* m_typedConfigurations;
02300     base::type::stringstream_t m_stream;
02301     std::string m_parentApplicationName;
02302     bool m_isConfigured;
02303     Configurations m_configurations;
02304     std::unordered_map<Level, unsigned int> m_unflushedCount;
02305     base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;
02306     LogBuilderPtr m_logBuilder;
02307
02308     friend class el::LogMessage;
02309     friend class el::Loggers;
02310     friend class el::Helpers;
02311     friend class el::base::RegisteredLoggers;
02312     friend class el::base::DefaultLogDispatchCallback;
02313     friend class el::base::MessageBuilder;
02314     friend class el::base::Writer;
02315     friend class el::base::PErrorWriter;
02316     friend class el::base::Storage;
02317     friend class el::base::PerformanceTracker;
02318     friend class el::base::LogDispatcher;
02319
02320     Logger(void);
02321
02322 #if ELPP_VARIADIC_TEMPLATES_SUPPORTED
02323     template <typename T, typename... Args>
02324     void log_(Level, int, const char*, const T&, const Args&...);
02325
02326     template <typename T>
02327     inline void log_(Level, int, const T&);
02328
02329     template <typename T, typename... Args>
02330     void log(Level, const char*, const T&, const Args&...);
02331
02332     template <typename T>
02333     inline void log(Level, const T&);
02334 #endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
02335
02336     void initUnflushedCount(void);
02337
02338     inline base::type::stringstream_t& stream(void) {
02339         return m_stream;
02340     }

```

```

02341     }
02342
02343     void resolveLoggerFormatSpec(void) const;
02344 };
02345 namespace base {
02346 class RegisteredLoggers : public base::utils::Registry<Logger, std::string> {
02347 public:
02348     explicit RegisteredLoggers(const LogBuilderPtr& defaultLogBuilder);
02349
02350     virtual ~RegisteredLoggers(void) {
02351         unsafeFlushAll();
02352     }
02353
02354     inline void setDefaultConfigurations(const Configurations& configurations) {
02355         base::threading::ScopedLock scopedLock(lock());
02356         m_defaultConfigurations.setFromBase(const_cast<Configurations*>(&configurations));
02357     }
02358
02359     inline Configurations* defaultConfigurations(void) {
02360         return &m_defaultConfigurations;
02361     }
02362
02363     Logger* get(const std::string& id, bool forceCreation = true);
02364
02365     template <typename T>
02366     inline bool installLoggerRegistrationCallback(const std::string& id) {
02367         return base::utils::Utils::installCallback<T, base::type::LoggerRegistrationCallbackPtr>(id,
02368             &m_loggerRegistrationCallbacks);
02369     }
02370
02371     template <typename T>
02372     inline void uninstallLoggerRegistrationCallback(const std::string& id) {
02373         return base::utils::Utils::uninstallCallback<T, base::type::LoggerRegistrationCallbackPtr>(id,
02374             &m_loggerRegistrationCallbacks);
02375     }
02376
02377     template <typename T>
02378     inline T* loggerRegistrationCallback(const std::string& id) {
02379         return base::utils::Utils::callback<T, base::type::LoggerRegistrationCallbackPtr>(id,
02380             &m_loggerRegistrationCallbacks);
02381     }
02382
02383     bool remove(const std::string& id);
02384
02385     inline bool has(const std::string& id) {
02386         return get(id, false) != nullptr;
02387     }
02388
02389     inline void unregister(Logger*& logger) {
02390         base::threading::ScopedLock scopedLock(lock());
02391         base::utils::Registry<Logger, std::string>::unregister(logger->id());
02392     }
02393
02394     inline LogStreamsReferenceMapPtr logStreamsReference(void) {
02395         return m_logStreamsReference;
02396     }
02397
02398     inline void flushAll(void) {
02399         base::threading::ScopedLock scopedLock(lock());
02400         unsafeFlushAll();
02401     }
02402
02403     inline void setDefaultLogBuilder(LogBuilderPtr& logBuilderPtr) {
02404         base::threading::ScopedLock scopedLock(lock());
02405         m_defaultLogBuilder = logBuilderPtr;
02406     }
02407 private:
02408     LogBuilderPtr m_defaultLogBuilder;
02409     Configurations m_defaultConfigurations;
02410     base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;
02411     std::unordered_map<std::string, base::type::LoggerRegistrationCallbackPtr>
02412     m_loggerRegistrationCallbacks;
02413     friend class el::base::Storage;
02414
02415     void unsafeFlushAll(void);
02416 };
02417 class VRegistry : base::NoCopy, public base::threading::ThreadSafe {
02418 public:
02419     explicit VRegistry(base::type::VerboseLevel level, base::type::EnumType* pFlags);
02420
02421     void setLevel(base::type::VerboseLevel level);
02422
02423     inline base::type::VerboseLevel level(void) const {
02424         return m_level;
02425     }
02426
02427

```

```

02428 inline void clearModules(void) {
02429     base::threading::ScopedLock scopedLock(lock());
02430     m_modules.clear();
02431 }
02432
02433 void setModules(const char* modules);
02434
02435 bool allowed(base::type::VerboseLevel vlevel, const char* file);
02436
02437 inline const std::unordered_map<std::string, base::type::VerboseLevel>& modules(void) const {
02438     return m_modules;
02439 }
02440
02441 void setFromArgs(const base::utils::CommandLineArgs* commandLineArgs);
02442
02443 inline bool vModulesEnabled(void) {
02444     return !base::utils::hasFlag(LoggingFlag::DisableVModules, *m_pFlags);
02445 }
02446
02447 private:
02448     base::type::VerboseLevel m_level;
02449     base::type::EnumType* m_pFlags;
02450     std::unordered_map<std::string, base::type::VerboseLevel> m_modules;
02451 };
02452
02453 // namespace base
02454 class LogMessage {
02455 public:
02456     LogMessage(Level level, const std::string& file, base::type::LineNumber line, const std::string&
02457 func,
02458         base::type::VerboseLevel verboseLevel, Logger* logger) :
02459         m_level(level), m_file(file), m_line(line), m_func(func),
02460         m_verboseLevel(verboseLevel), m_logger(logger), m_message(logger->stream().str()) {
02461 }
02462 inline Level level(void) const {
02463     return m_level;
02464 }
02465 inline const std::string& file(void) const {
02466     return m_file;
02467 }
02468 inline base::type::LineNumber line(void) const {
02469     return m_line;
02470 }
02471 inline const std::string& func(void) const {
02472     return m_func;
02473 }
02474 inline base::type::VerboseLevel verboseLevel(void) const {
02475     return m_verboseLevel;
02476 }
02477 inline Logger* logger(void) const {
02478     return m_logger;
02479 }
02480 inline const base::type::string_t& message(void) const {
02481     return m_message;
02482 }
02483 private:
02484     Level m_level;
02485     std::string m_file;
02486     base::type::LineNumber m_line;
02487     std::string m_func;
02488     base::type::VerboseLevel m_verboseLevel;
02489     Logger* m_logger;
02490     base::type::string_t m_message;
02491 };
02492 namespace base {
02493 #if ELPP_ASYNC_LOGGING
02494 class AsyncLogItem {
02495 public:
02496     explicit AsyncLogItem(const LogMessage& logMessage, const LogDispatchData& data, const
02497 base::type::string_t& logLine)
02498         : m_logMessage(logMessage), m_dispatchData(data), m_logLine(logLine) {}
02499     virtual ~AsyncLogItem() {}
02500     inline LogMessage* logMessage(void) {
02501         return &m_logMessage;
02502     }
02503     inline LogDispatchData* data(void) {
02504         return &m_dispatchData;
02505     }
02506     inline base::type::string_t logLine(void) {
02507         return m_logLine;
02508     }
02509 private:
02510     LogMessage m_logMessage;
02511     LogDispatchData m_dispatchData;
02512     base::type::string_t m_logLine;
02513 };
02514 class AsyncLogQueue : public base::threading::ThreadSafe {
02515 public:

```

```

02514     virtual ~AsyncLogQueue() {
02515         ELPP_INTERNAL_INFO(6, "~AsyncLogQueue");
02516     }
02517
02518     inline AsyncLogItem next(void) {
02519         base::threading::ScopedLock scopedLock(lock());
02520         AsyncLogItem result = m_queue.front();
02521         m_queue.pop();
02522         return result;
02523     }
02524
02525     inline void push(const AsyncLogItem& item) {
02526         base::threading::ScopedLock scopedLock(lock());
02527         m_queue.push(item);
02528     }
02529     inline void pop(void) {
02530         base::threading::ScopedLock scopedLock(lock());
02531         m_queue.pop();
02532     }
02533     inline AsyncLogItem front(void) {
02534         base::threading::ScopedLock scopedLock(lock());
02535         return m_queue.front();
02536     }
02537     inline bool empty(void) {
02538         base::threading::ScopedLock scopedLock(lock());
02539         return m_queue.empty();
02540     }
02541 private:
02542     std::queue<AsyncLogItem> m_queue;
02543 };
02544 class IWorker {
02545 public:
02546     virtual ~IWorker() {}
02547     virtual void start() = 0;
02548 };
02549 #endif // ELPP_ASYNC_LOGGING
02551 class Storage : base::NoCopy, public base::threading::ThreadSafe {
02552 public:
02553     #if ELPP_ASYNC_LOGGING
02554         Storage(const LogBuilderPtr& defaultLogBuilder, base::IWorker* asyncDispatchWorker);
02555     #else
02556         explicit Storage(const LogBuilderPtr& defaultLogBuilder);
02557     #endif // ELPP_ASYNC_LOGGING
02558
02559     virtual ~Storage(void);
02560
02561     inline bool validateEveryNCounter(const char* filename, base::type::LineNumber lineNumber,
std::size_t occasion) {
02562         return hitCounters()->validateEveryN(filename, lineNumber, occasion);
02563     }
02564
02565     inline bool validateAfterNCounter(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
02566         return hitCounters()->validateAfterN(filename, lineNumber, n);
02567     }
02568
02569     inline bool validateNTimesCounter(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
02570         return hitCounters()->validateNTimes(filename, lineNumber, n);
02571     }
02572
02573     inline base::RegisteredHitCounters* hitCounters(void) const {
02574         return m_registeredHitCounters;
02575     }
02576
02577     inline base::RegisteredLoggers* registeredLoggers(void) const {
02578         return m_registeredLoggers;
02579     }
02580
02581     inline base::VRegistry* vRegistry(void) const {
02582         return m_vRegistry;
02583     }
02584
02585     #if ELPP_ASYNC_LOGGING
02586         inline base::AsyncLogQueue* asyncLogQueue(void) const {
02587             return m_asyncLogQueue;
02588         }
02589     #endif // ELPP_ASYNC_LOGGING
02590
02591     inline const base::utils::CommandLineArgs* commandLineArgs(void) const {
02592         return &m_commandLineArgs;
02593     }
02594
02595     inline void addFlag(LoggingFlag flag) {
02596         base::utils::addFlag(flag, &m_flags);
02597     }
02598
02599

```

```

02599 inline void removeFlag(LoggingFlag flag) {
02600     base::utils::removeFlag(flag, &m_flags);
02601 }
02602
02603 inline bool hasFlag(LoggingFlag flag) const {
02604     return base::utils::hasFlag(flag, m_flags);
02605 }
02606
02607 inline base::type::EnumType flags(void) const {
02608     return m_flags;
02609 }
02610
02611 inline void setFlags(base::type::EnumType flags) {
02612     m_flags = flags;
02613 }
02614
02615 inline void setPreRollOutCallback(const PreRollOutCallback& callback) {
02616     m_preRollOutCallback = callback;
02617 }
02618
02619 inline void unsetPreRollOutCallback(void) {
02620     m_preRollOutCallback = base::defaultPreRollOutCallback;
02621 }
02622
02623 inline PreRollOutCallback& preRollOutCallback(void) {
02624     return m_preRollOutCallback;
02625 }
02626
02627 bool hasCustomFormatSpecifier(const char* formatSpecifier);
02628 void installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier);
02629 bool uninstallCustomFormatSpecifier(const char* formatSpecifier);
02630
02631 const std::vector<CustomFormatSpecifier>* customFormatSpecifiers(void) const {
02632     return &m_customFormatSpecifiers;
02633 }
02634
02635 base::threading::Mutex& customFormatSpecifiersLock() {
02636     return m_customFormatSpecifiersLock;
02637 }
02638
02639 inline void setLoggingLevel(Level level) {
02640     m_loggingLevel = level;
02641 }
02642
02643 template <typename T>
02644 inline bool installLogDispatchCallback(const std::string& id) {
02645     return base::utils::Utils::installCallback<T, base::type::LogDispatchCallbackPtr>(id,
02646 &m_logDispatchCallbacks);
02647 }
02648
02649 template <typename T>
02650 inline void uninstallLogDispatchCallback(const std::string& id) {
02651     base::utils::Utils::uninstallCallback<T, base::type::LogDispatchCallbackPtr>(id,
02652 &m_logDispatchCallbacks);
02653 }
02654
02655 template <typename T>
02656 inline T* logDispatchCallback(const std::string& id) {
02657     return base::utils::Utils::callback<T, base::type::LogDispatchCallbackPtr>(id,
02658 &m_logDispatchCallbacks);
02659 }
02660
02661 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02662 template <typename T>
02663 inline bool installPerformanceTrackingCallback(const std::string& id) {
02664     return base::utils::Utils::installCallback<T, base::type::PerformanceTrackingCallbackPtr>(id,
02665 &m_performanceTrackingCallbacks);
02666 }
02667
02668 template <typename T>
02669 inline void uninstallPerformanceTrackingCallback(const std::string& id) {
02670     base::utils::Utils::uninstallCallback<T, base::type::PerformanceTrackingCallbackPtr>(id,
02671 &m_performanceTrackingCallbacks);
02672 }
02673
02674 template <typename T>
02675 inline T* performanceTrackingCallback(const std::string& id) {
02676     return base::utils::Utils::callback<T, base::type::PerformanceTrackingCallbackPtr>(id,
02677 &m_performanceTrackingCallbacks);
02678 }
02679 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
02680
02681 inline void setThreadName(const std::string& name) {
02682     if (name.empty()) return;
02683     base::threading::ScopedLock scopedLock(m_threadNamesLock);
02684     m_threadNames[base::threading::getCurrentThreadId()] = name;
02685 }
02686

```

```

02683 inline std::string getThreadName(const std::string& threadId) {
02684     base::threading::ScopedLock scopedLock(m_threadNamesLock);
02685     std::unordered_map<std::string, std::string>::const_iterator it = m_threadNames.find(threadId);
02686     if (it == m_threadNames.end()) {
02687         return threadId;
02688     }
02689     return it->second;
02690 }
02691 private:
02692 base::RegisteredHitCounters* m_registeredHitCounters;
02693 base::RegisteredLoggers* m_registeredLoggers;
02694 base::type::EnumType m_flags;
02695 base::VRegistry* m_vRegistry;
02696 #if ELPP_ASYNC_LOGGING
02697 base::AsyncLogQueue* m_asyncLogQueue;
02698 base::IWorker* m_asyncDispatchWorker;
02699 #endif // ELPP_ASYNC_LOGGING
02700 base::utils::CommandLineArgs m_commandLineArgs;
02701 PreRollOutCallback m_preRollOutCallback;
02702 std::unordered_map<std::string, base::type::LogDispatchCallbackPtr> m_logDispatchCallbacks;
02703 std::unordered_map<std::string, base::type::PerformanceTrackingCallbackPtr>
m_performanceTrackingCallbacks;
02704 std::unordered_map<std::string, std::string> m_threadNames;
02705 std::vector<CustomFormatSpecifier> m_customFormatSpecifiers;
02706 base::threading::Mutex m_customFormatSpecifiersLock;
02707 base::threading::Mutex m_threadNamesLock;
02708 Level m_loggingLevel;
02709
02710 friend class el::Helpers;
02711 friend class el::base::DefaultLogDispatchCallback;
02712 friend class el::LogBuilder;
02713 friend class el::base::MessageBuilder;
02714 friend class el::base::Writer;
02715 friend class el::base::PerformanceTracker;
02716 friend class el::base::LogDispatcher;
02717
02718 void setApplicationArguments(int argc, char** argv);
02719
02720 inline void setApplicationArguments(int argc, const char** argv) {
02721     setApplicationArguments(argc, const_cast<char**>(argv));
02722 }
02723 };
02724 extern ELPP_EXPORT base::type::StoragePointer elStorage;
02725 #define ELPP el::base::elStorage
02726 class DefaultLogDispatchCallback : public LogDispatchCallback {
02727 protected:
02728     void handle(const LogDispatchData* data);
02729 private:
02730     const LogDispatchData* m_data;
02731     void dispatch(base::type::string_t&& logLine);
02732 };
02733 #if ELPP_ASYNC_LOGGING
02734 class AsyncLogDispatchCallback : public LogDispatchCallback {
02735 protected:
02736     void handle(const LogDispatchData* data);
02737 };
02738 class AsyncDispatchWorker : public base::IWorker, public base::threading::ThreadSafe {
02739 public:
02740     AsyncDispatchWorker();
02741     virtual ~AsyncDispatchWorker();
02742
02743     bool clean(void);
02744     void emptyQueue(void);
02745     virtual void start(void);
02746     void handle(AsyncLogItem* logItem);
02747     void run(void);
02748
02749     void setContinueRunning(bool value) {
02750         base::threading::ScopedLock scopedLock(m_continueRunningLock);
02751         m_continueRunning = value;
02752     }
02753
02754     bool continueRunning(void) const {
02755         return m_continueRunning;
02756     }
02757 private:
02758     std::condition_variable cv;
02759     bool m_continueRunning;
02760     base::threading::Mutex m_continueRunningLock;
02761 };
02762 #endif // ELPP_ASYNC_LOGGING
02763 } // namespace base
02764 namespace base {
02765 class DefaultLogBuilder : public LogBuilder {
02766 public:
02767     base::type::string_t build(const LogMessage* logMessage, bool appendNewLine) const;
02768 };

```



```

02770 class LogDispatcher : base::NoCopy {
02771 public:
02772     LogDispatcher(bool proceed, LogMessage* logMessage, base::DispatchAction dispatchAction) :
02773         m_proceed(proceed),
02774         m_logMessage(logMessage),
02775         m_dispatchAction(std::move(dispatchAction)) {
02776     }
02777
02778     void dispatch(void);
02779
02780 private:
02781     bool m_proceed;
02782     LogMessage* m_logMessage;
02783     base::DispatchAction m_dispatchAction;
02784 };
02785 #if defined(ELPP_STL_LOGGING)
02792 namespace workarounds {
02794     template <typename T, typename Container>
02795     class IterableContainer {
02796     public:
02797         typedef typename Container::iterator iterator;
02798         typedef typename Container::const_iterator const_iterator;
02799         IterableContainer(void) {}
02800         virtual ~IterableContainer(void) {}
02801         iterator begin(void) {
02802             return getContainer().begin();
02803         }
02804         iterator end(void) {
02805             return getContainer().end();
02806         }
02807     private:
02808         virtual Container& getContainer(void) = 0;
02809     };
02811     template<typename T, typename Container = std::vector<T>, typename Comparator = std::less<typename
Container::value_type>
02812     class IterablePriorityQueue : public IterableContainer<T, Container>,
02813     public std::priority_queue<T, Container, Comparator> {
02814     public:
02815         IterablePriorityQueue(std::priority_queue<T, Container, Comparator> queue_) {
02816             std::size_t count_ = 0;
02817             while (++count_ < base::consts::kMaxLogPerContainer && !queue_.empty()) {
02818                 this->push(queue_.top());
02819                 queue_.pop();
02820             }
02821         }
02822     private:
02823         inline Container& getContainer(void) {
02824             return this->c;
02825         }
02826     };
02828     template<typename T, typename Container = std::deque<T>
02829     class IterableQueue : public IterableContainer<T, Container>, public std::queue<T, Container> {
02830     public:
02831         IterableQueue(std::queue<T, Container> queue_) {
02832             std::size_t count_ = 0;
02833             while (++count_ < base::consts::kMaxLogPerContainer && !queue_.empty()) {
02834                 this->push(queue_.front());
02835                 queue_.pop();
02836             }
02837         }
02838     private:
02839         inline Container& getContainer(void) {
02840             return this->c;
02841         }
02842     };
02844     template<typename T, typename Container = std::deque<T>
02845     class IterableStack : public IterableContainer<T, Container>, public std::stack<T, Container> {
02846     public:
02847         IterableStack(std::stack<T, Container> stack_) {
02848             std::size_t count_ = 0;
02849             while (++count_ < base::consts::kMaxLogPerContainer && !stack_.empty()) {
02850                 this->push(stack_.top());
02851                 stack_.pop();
02852             }
02853         }
02854     private:
02855         inline Container& getContainer(void) {
02856             return this->c;
02857         }
02858     };
02859 } // namespace workarounds
02860 #endif // defined(ELPP_STL_LOGGING)
02861 // Log message builder
02862 class MessageBuilder {
02863 public:
02864     MessageBuilder(void) : m_logger(nullptr), m_containerLogSeparator(ELPP_LITERAL("")) {}
02865     void initialize(Logger* logger);

```

```

02866
02867 # define ELPP_SIMPLE_LOG(LOG_TYPE)\
02868 MessageBuilder& operator«(LOG_TYPE msg) {\
02869 m_logger->stream() « msg;\
02870 if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {\
02871 m_logger->stream() « " ";\
02872 }\
02873 return *this;\
02874 }
02875
02876 inline MessageBuilder& operator«(const std::string& msg) {
02877     return operator«(msg.c_str());
02878 }
02879 ELPP_SIMPLE_LOG(char)
02880 ELPP_SIMPLE_LOG(bool)
02881 ELPP_SIMPLE_LOG(signed short)
02882 ELPP_SIMPLE_LOG(unsigned short)
02883 ELPP_SIMPLE_LOG(signed int)
02884 ELPP_SIMPLE_LOG(unsigned int)
02885 ELPP_SIMPLE_LOG(signed long)
02886 ELPP_SIMPLE_LOG(unsigned long)
02887 ELPP_SIMPLE_LOG(float)
02888 ELPP_SIMPLE_LOG(double)
02889 ELPP_SIMPLE_LOG(char*)
02890 ELPP_SIMPLE_LOG(const char*)
02891 ELPP_SIMPLE_LOG(const void*)
02892 ELPP_SIMPLE_LOG(long double)
02893 inline MessageBuilder& operator«(const std::wstring& msg) {
02894     return operator«(msg.c_str());
02895 }
02896 MessageBuilder& operator«(const wchar_t* msg);
02897 // ostream manipulators
02898 inline MessageBuilder& operator«(std::ostream& (*OStreamMani) (std::ostream&)) {
02899     m_logger->stream() « OStreamMani;
02900     return *this;
02901 }
02902 #define ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(temp)
02903 template <typename T>
02904 inline MessageBuilder& operator«(const temp<T>& template_inst) {
02905     return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());
02906 }
02907 #define ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(temp)
02908 template <typename T1, typename T2>
02909 inline MessageBuilder& operator«(const temp<T1, T2>& template_inst) {
02910     return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());
02911 }
02912 #define ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(temp)
02913 template <typename T1, typename T2, typename T3>
02914 inline MessageBuilder& operator«(const temp<T1, T2, T3>& template_inst) {
02915     return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());
02916 }
02917 #define ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(temp)
02918 template <typename T1, typename T2, typename T3, typename T4>
02919 inline MessageBuilder& operator«(const temp<T1, T2, T3, T4>& template_inst) {
02920     return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());
02921 }
02922 #define ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(temp)
02923 template <typename T1, typename T2, typename T3, typename T4, typename T5>
02924 inline MessageBuilder& operator«(const temp<T1, T2, T3, T4, T5>& template_inst) {
02925     return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());
02926 }
02927
02928 #if defined(ELPP_STL_LOGGING)
02929 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::vector)
02930 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::list)
02931 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::deque)
02932 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(std::set)
02933 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(std::multiset)
02934 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::map)
02935 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::multimap)
02936 template <class T, class Container>
02937 inline MessageBuilder& operator«(const std::queue<T, Container>& queue_) {
02938     base::workarounds::IterableQueue<T, Container> iterableQueue_ =
02939         static_cast<base::workarounds::IterableQueue<T, Container>>(queue_);
02940     return writeIterator(iterableQueue_.begin(), iterableQueue_.end(), iterableQueue_.size());
02941 }
02942 template <class T, class Container>
02943 inline MessageBuilder& operator«(const std::stack<T, Container>& stack_) {
02944     base::workarounds::IterableStack<T, Container> iterableStack_ =
02945         static_cast<base::workarounds::IterableStack<T, Container>>(stack_);
02946     return writeIterator(iterableStack_.begin(), iterableStack_.end(), iterableStack_.size());
02947 }
02948 template <class T, class Container, class Comparator>
02949 inline MessageBuilder& operator«(const std::priority_queue<T, Container, Comparator>&
priorityQueue_) {
02950     base::workarounds::IterablePriorityQueue<T, Container, Comparator> iterablePriorityQueue_ =
02951         static_cast<base::workarounds::IterablePriorityQueue<T, Container, Comparator>

```

```

>(priorityQueue_);
02952     return writeIterator(iterablePriorityQueue_.begin(), iterablePriorityQueue_.end(),
iterablePriorityQueue_.size());
02953 }
02954 template <class First, class Second>
02955 MessageBuilder& operator<<(const std::pair<First, Second>& pair_) {
02956     m_logger->stream() << ELPP_LITERAL("(");
02957     operator << (static_cast<First>(pair_.first));
02958     m_logger->stream() << ELPP_LITERAL(", ");
02959     operator << (static_cast<Second>(pair_.second));
02960     m_logger->stream() << ELPP_LITERAL(")");
02961     return *this;
02962 }
02963 template <std::size_t Size>
02964 MessageBuilder& operator<<(const std::bitset<Size>& bitset_) {
02965     m_logger->stream() << ELPP_LITERAL("[");
02966     operator << (bitset_.to_string());
02967     m_logger->stream() << ELPP_LITERAL("]");
02968     return *this;
02969 }
02970 # if defined(ELPP_LOG_STD_ARRAY)
02971 template <class T, std::size_t Size>
02972 inline MessageBuilder& operator<<(const std::array<T, Size>& array) {
02973     return writeIterator(array.begin(), array.end(), array.size());
02974 }
02975 # endif // defined(ELPP_LOG_STD_ARRAY)
02976 # if defined(ELPP_LOG_UNORDERED_MAP)
02977 ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(std::unordered_map)
02978 ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(std::unordered_multimap)
02979 # endif // defined(ELPP_LOG_UNORDERED_MAP)
02980 # if defined(ELPP_LOG_UNORDERED_SET)
02981 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::unordered_set)
02982 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::unordered_multiset)
02983 # endif // defined(ELPP_LOG_UNORDERED_SET)
02984 #endif // defined(ELPP_STL_LOGGING)
02985 #if defined(ELPP_QT_LOGGING)
02986 inline MessageBuilder& operator<<(const QString& msg) {
02987     # if defined(ELPP_UNICODE)
02988     m_logger->stream() << msg.toStdWString();
02989     # else
02990     m_logger->stream() << msg.toStdString();
02991     # endif // defined(ELPP_UNICODE)
02992     return *this;
02993 }
02994 inline MessageBuilder& operator<<(const QByteArray& msg) {
02995     return operator << (QString(msg));
02996 }
02997 inline MessageBuilder& operator<<(const QStringRef& msg) {
02998     return operator<<(msg.toString());
02999 }
03000 inline MessageBuilder& operator<<(qint64 msg) {
03001     # if defined(ELPP_UNICODE)
03002     m_logger->stream() << QString::number(msg).toStdWString();
03003     # else
03004     m_logger->stream() << QString::number(msg).toStdString();
03005     # endif // defined(ELPP_UNICODE)
03006     return *this;
03007 }
03008 inline MessageBuilder& operator<<(quint64 msg) {
03009     # if defined(ELPP_UNICODE)
03010     m_logger->stream() << QString::number(msg).toStdWString();
03011     # else
03012     m_logger->stream() << QString::number(msg).toStdString();
03013     # endif // defined(ELPP_UNICODE)
03014     return *this;
03015 }
03016 inline MessageBuilder& operator<<(QChar msg) {
03017     m_logger->stream() << msg.toLatin1();
03018     return *this;
03019 }
03020 inline MessageBuilder& operator<<(const QLatin1String& msg) {
03021     m_logger->stream() << msg.toLatin1();
03022     return *this;
03023 }
03024 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QList)
03025 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QVector)
03026 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QQueue)
03027 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QSet)
03028 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QLinkedList)
03029 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QStack)
03030 template <typename First, typename Second>
03031 MessageBuilder& operator<<(const QMap<First, Second>& pair_) {
03032     m_logger->stream() << ELPP_LITERAL("(");
03033     operator << (static_cast<First>(pair_.first));
03034     m_logger->stream() << ELPP_LITERAL(", ");
03035     operator << (static_cast<Second>(pair_.second));
03036     m_logger->stream() << ELPP_LITERAL(")");

```

```

03037     return *this;
03038 }
03039 template <typename K, typename V>
03040 MessageBuilder& operator<<(const QMap<K, V>& map_) {
03041     m_logger->stream() << ELPP_LITERAL("[");
03042     QList<K> keys = map_.keys();
03043     typename QList<K>::const_iterator begin = keys.begin();
03044     typename QList<K>::const_iterator end = keys.end();
03045     int max_ = static_cast<int>(base::consts::kMaxLogPerContainer); // to prevent warning
03046     for (int index_ = 0; begin != end && index_ < max_; ++index_, ++begin) {
03047         m_logger->stream() << ELPP_LITERAL("(");
03048         operator << (static_cast<K>)(*begin));
03049         m_logger->stream() << ELPP_LITERAL(", ");
03050         operator << (static_cast<V>(map_.value(*begin)));
03051         m_logger->stream() << ELPP_LITERAL(" ");
03052         m_logger->stream() << ((index_ < keys.size() - 1) ? m_containerLogSeparator : ELPP_LITERAL(""));
03053     }
03054     if (begin != end) {
03055         m_logger->stream() << ELPP_LITERAL("...");
03056     }
03057     m_logger->stream() << ELPP_LITERAL("]");
03058     return *this;
03059 }
03060 template <typename K, typename V>
03061 inline MessageBuilder& operator<<(const QMultiMap<K, V>& map_) {
03062     operator << (static_cast<QMap<K, V>>(map_));
03063     return *this;
03064 }
03065 template <typename K, typename V>
03066 MessageBuilder& operator<<(const QHash<K, V>& hash_) {
03067     m_logger->stream() << ELPP_LITERAL("[");
03068     QList<K> keys = hash_.keys();
03069     typename QList<K>::const_iterator begin = keys.begin();
03070     typename QList<K>::const_iterator end = keys.end();
03071     int max_ = static_cast<int>(base::consts::kMaxLogPerContainer); // prevent type warning
03072     for (int index_ = 0; begin != end && index_ < max_; ++index_, ++begin) {
03073         m_logger->stream() << ELPP_LITERAL("(");
03074         operator << (static_cast<K>(*begin));
03075         m_logger->stream() << ELPP_LITERAL(", ");
03076         operator << (static_cast<V>(hash_.value(*begin)));
03077         m_logger->stream() << ELPP_LITERAL(" ");
03078         m_logger->stream() << ((index_ < keys.size() - 1) ? m_containerLogSeparator : ELPP_LITERAL(""));
03079     }
03080     if (begin != end) {
03081         m_logger->stream() << ELPP_LITERAL("...");
03082     }
03083     m_logger->stream() << ELPP_LITERAL("]");
03084     return *this;
03085 }
03086 template <typename K, typename V>
03087 inline MessageBuilder& operator<<(const QMultiHash<K, V>& multiHash_) {
03088     operator << (static_cast<QHash<K, V>>(multiHash_));
03089     return *this;
03090 }
03091 #endif // defined(ELPP_QT_LOGGING)
03092 #if defined(ELPP_BOOST_LOGGING)
03093 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::vector)
03094 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::stable_vector)
03095 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::list)
03096 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::deque)
03097 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(boost::container::map)
03098 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(boost::container::flat_map)
03099 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(boost::container::set)
03100 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(boost::container::flat_set)
03101 #endif // defined(ELPP_BOOST_LOGGING)
03102
03111 #define MAKE_CONTAINERELPP_FRIENDLY(ContainerType, SizeMethod, ElementInstance) \
03112 el::base::type::ostream_t& operator<<(el::base::type::ostream_t& ss, const ContainerType& container) {\
03113     const el::base::type::char_t* sep = ELPP->hasFlag(el::LoggingFlag::NewLineForContainer) ? \
03114     ELPP_LITERAL("\n    ") : ELPP_LITERAL(", ");\
03115     ContainerType::const_iterator elem = container.begin();\
03116     ContainerType::const_iterator endElem = container.end();\
03117     std::size_t size_ = container.SizeMethod(); \
03118     ss << ELPP_LITERAL("[");\
03119     for (std::size_t i = 0; elem != endElem && i < el::base::consts::kMaxLogPerContainer; ++i, ++elem) { \
03120         ss << ElementInstance;\
03121         ss << ((i < size_ - 1) ? sep : ELPP_LITERAL("")); \
03122     }\
03123     if (elem != endElem) {\
03124         ss << ELPP_LITERAL("...");\
03125     }\
03126     ss << ELPP_LITERAL("]");\
03127     return ss;\
03128 }
03129 #if defined(ELPP_WXWIDGETS_LOGGING)
03130 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(wxVector)
03131 # define ELPP_WX_PTR_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(),

```

```

    (*elem))
03132 # define ELPP_WX_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(), (*elem))
03133 # define ELPP_WX_HASH_MAP_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(), \
03134 ELPP_LITERAL("(") « elem->first « ELPP_LITERAL(", ") « elem->second « ELPP_LITERAL(")")
03135 #else
03136 # define ELPP_WX_PTR_ENABLED(ContainerType)
03137 # define ELPP_WX_ENABLED(ContainerType)
03138 # define ELPP_WX_HASH_MAP_ENABLED(ContainerType)
03139 #endif // defined(ELPP_WXWIDGETS_LOGGING)
03140 // Other classes
03141 template <class Class>
03142 ELPP_SIMPLE_LOG(const Class&)
03143 #undef ELPP_SIMPLE_LOG
03144 #undef ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG
03145 #undef ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG
03146 #undef ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG
03147 #undef ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG
03148 #undef ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG
03149 private:
03150 Logger* m_logger;
03151 const base::type::char_t* m_containerLogSeparator;
03152
03153 template<class Iterator>
03154 MessageBuilder& writeIterator(Iterator begin_, Iterator end_, std::size_t size_) {
03155     m_logger->stream() « ELPP_LITERAL("[");
03156     for (std::size_t i = 0; begin_ != end_ && i < base::consts::kMaxLogPerContainer; ++i, ++begin_) {
03157         operator « (*begin_);
03158         m_logger->stream() « ((i < size_ - 1) ? m_containerLogSeparator : ELPP_LITERAL(""));
03159     }
03160     if (begin_ != end_) {
03161         m_logger->stream() « ELPP_LITERAL("...");
03162     }
03163     m_logger->stream() « ELPP_LITERAL("]");
03164     if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {
03165         m_logger->stream() « " ";
03166     }
03167     return *this;
03168 }
03169 };
03170 class NullWriter : base::NoCopy {
03171 public:
03172     NullWriter(void) {}
03173
03174     // Null manipulator
03175     inline NullWriter& operator<<(std::ostream& (*)(std::ostream&)) {
03176         return *this;
03177     }
03178
03179     template <typename T>
03180     inline NullWriter& operator<<(const T&) {
03181         return *this;
03182     }
03183
03184     inline operator bool() {
03185         return true;
03186     }
03187 };
03188 };
03189 class Writer : base::NoCopy {
03190 public:
03191     Writer(Level level, const char* file, base::type::LineNumber line,
03192            const char* func, base::DispatchAction dispatchAction = base::DispatchAction::NormalLog,
03193            base::type::VerboseLevel verboseLevel = 0) :
03194         m_msg(nullptr), m_level(level), m_file(file), m_line(line), m_func(func),
03195         m_verboseLevel(verboseLevel),
03196         m_logger(nullptr), m_proceed(false), m_dispatchAction(dispatchAction) {
03197     }
03198
03199     Writer(LogMessage* msg, base::DispatchAction dispatchAction = base::DispatchAction::NormalLog) :
03200         m_msg(msg), m_level(msg != nullptr ? msg->level() : Level::Unknown),
03201         m_line(0), m_logger(nullptr), m_proceed(false), m_dispatchAction(dispatchAction) {
03202     }
03203
03204     virtual ~Writer(void) {
03205         processDispatch();
03206     }
03207
03208     template <typename T>
03209     inline Writer& operator<<(const T& log) {
03210 #if ELPP_LOGGING_ENABLED
03211         if (m_proceed) {
03212             m_messageBuilder « log;
03213         }
03214 #endif // ELPP_LOGGING_ENABLED
03215         return *this;
03216     }
03217
03218     inline Writer& operator<<(std::ostream& (*log)(std::ostream&)) {

```

```

03219 #if ELPP_LOGGING_ENABLED
03220     if (m_proceed) {
03221         m_messageBuilder « log;
03222     }
03223 #endif // ELPP_LOGGING_ENABLED
03224     return *this;
03225 }
03226
03227 inline operator bool() {
03228     return true;
03229 }
03230
03231 Writer& construct(Logger* logger, bool needLock = true);
03232 Writer& construct(int count, const char* loggerIds, ...);
03233 protected:
03234 LogMessage* m_msg;
03235 Level m_level;
03236 const char* m_file;
03237 const base::type::LineNumber m_line;
03238 const char* m_func;
03239 base::type::VerboseLevel m_verboseLevel;
03240 Logger* m_logger;
03241 bool m_proceed;
03242 base::MessageBuilder m_messageBuilder;
03243 base::DispatchAction m_dispatchAction;
03244 std::vector<std::string> m_loggerIds;
03245 friend class el::Helpers;
03246
03247 void initializeLogger(const std::string& loggerId, bool lookup = true, bool needLock = true);
03248 void processDispatch();
03249 void triggerDispatch(void);
03250 };
03251 class PErrorWriter : public base::Writer {
03252 public:
03253     PErrorWriter(Level level, const char* file, base::type::LineNumber line,
03254                 const char* func, base::DispatchAction dispatchAction =
03255                 base::DispatchAction::NormalLog,
03256                 base::type::VerboseLevel verboseLevel = 0) :
03257         base::Writer(level, file, line, func, dispatchAction, verboseLevel) {
03258     }
03259     virtual ~PErrorWriter(void);
03260 };
03261 } // namespace base
03262 // Logging from Logger class. Why this is here? Because we have Storage and Writer class available
03263 #if ELPP_VARIADIC_TEMPLATES_SUPPORTED
03264 template <typename T, typename... Args>
03265 void Logger::log_(Level level, int vlevel, const char* s, const T& value, const Args&... args) {
03266     base::MessageBuilder b;
03267     b.initialize(this);
03268     while (*s) {
03269         if (*s == base::consts::kFormatSpecifierChar) {
03270             if (*(s + 1) == base::consts::kFormatSpecifierChar) {
03271                 ++s;
03272             } else {
03273                 if (*(s + 1) == base::consts::kFormatSpecifierCharValue) {
03274                     ++s;
03275                     b « value;
03276                     log_(level, vlevel, ++s, args...);
03277                     return;
03278                 }
03279             }
03280         }
03281         b « *s++;
03282     }
03283     ELPP_INTERNAL_ERROR("Too many arguments provided. Unable to handle. Please provide more format
03284     specifiers", false);
03285 }
03286 template <typename T>
03287 void Logger::log_(Level level, int vlevel, const T& log) {
03288     if (level == Level::Verbose) {
03289         if (ELPP->vRegistry()->allowed(vlevel, __FILE__) {
03290             base::Writer(Level::Verbose, "FILE", 0, "FUNCTION",
03291                         base::DispatchAction::NormalLog, vlevel).construct(this, false) « log;
03292         } else {
03293             stream().str(ELPP_LITERAL(""));
03294             releaseLock();
03295         }
03296     } else {
03297         base::Writer(level, "FILE", 0, "FUNCTION").construct(this, false) « log;
03298     }
03299 }
03300 template <typename T, typename... Args>
03301 inline void Logger::log(Level level, const char* s, const T& value, const Args&... args) {
03302     acquireLock(); // released in Writer!
03303     log_(level, 0, s, value, args...);
03304 }

```

```

03304 template <typename T>
03305 inline void Logger::log(Level level, const T& log) {
03306     acquireLock(); // released in Writer!
03307     log_(level, 0, log);
03308 }
03309 # if ELPP_VERBOSE_LOG
03310 template <typename T, typename... Args>
03311 inline void Logger::verbose(int vlevel, const char* s, const T& value, const Args&... args) {
03312     acquireLock(); // released in Writer!
03313     log_(el::Level::Verbose, vlevel, s, value, args...);
03314 }
03315 template <typename T>
03316 inline void Logger::verbose(int vlevel, const T& log) {
03317     acquireLock(); // released in Writer!
03318     log_(el::Level::Verbose, vlevel, log);
03319 }
03320 # else
03321 template <typename T, typename... Args>
03322 inline void Logger::verbose(int, const char*, const T&, const Args&...) {
03323     return;
03324 }
03325 template <typename T>
03326 inline void Logger::verbose(int, const T&) {
03327     return;
03328 }
03329 # endif // ELPP_VERBOSE_LOG
03330 # define LOGGER_LEVEL_WRITERS(FUNCTION_NAME, LOG_LEVEL)\
03331 template <typename T, typename... Args>\
03332 inline void Logger::FUNCTION_NAME(const char* s, const T& value, const Args&... args) {\
03333     log(LOG_LEVEL, s, value, args...);\
03334 }\
03335 template <typename T>\
03336 inline void Logger::FUNCTION_NAME(const T& value) {\
03337     log(LOG_LEVEL, value);\
03338 }\
03339 # define LOGGER_LEVEL_WRITERS_DISABLED(FUNCTION_NAME, LOG_LEVEL)\
03340 template <typename T, typename... Args>\
03341 inline void Logger::FUNCTION_NAME(const char*, const T&, const Args&...) {\
03342     return;\
03343 }\
03344 template <typename T>\
03345 inline void Logger::FUNCTION_NAME(const T&) {\
03346     return;\
03347 }\
03348 # if ELPP_INFO_LOG
03349 LOGGER_LEVEL_WRITERS(info, Level::Info)
03350 # else
03351 LOGGER_LEVEL_WRITERS_DISABLED(info, Level::Info)
03352 # endif // ELPP_INFO_LOG
03353 # if ELPP_DEBUG_LOG
03354 LOGGER_LEVEL_WRITERS(debug, Level::Debug)
03355 # else
03356 LOGGER_LEVEL_WRITERS_DISABLED(debug, Level::Debug)
03357 # endif // ELPP_DEBUG_LOG
03358 # if ELPP_WARNING_LOG
03359 LOGGER_LEVEL_WRITERS(warn, Level::Warning)
03360 # else
03361 LOGGER_LEVEL_WRITERS_DISABLED(warn, Level::Warning)
03362 # endif // ELPP_WARNING_LOG
03363 # if ELPP_ERROR_LOG
03364 LOGGER_LEVEL_WRITERS(error, Level::Error)
03365 # else
03366 LOGGER_LEVEL_WRITERS_DISABLED(error, Level::Error)
03367 # endif // ELPP_ERROR_LOG
03368 # if ELPP_FATAL_LOG
03369 LOGGER_LEVEL_WRITERS(fatal, Level::Fatal)
03370 # else
03371 LOGGER_LEVEL_WRITERS_DISABLED(fatal, Level::Fatal)
03372 # endif // ELPP_FATAL_LOG
03373 # if ELPP_TRACE_LOG
03374 LOGGER_LEVEL_WRITERS(trace, Level::Trace)
03375 # else
03376 LOGGER_LEVEL_WRITERS_DISABLED(trace, Level::Trace)
03377 # endif // ELPP_TRACE_LOG
03378 # undef LOGGER_LEVEL_WRITERS
03379 # undef LOGGER_LEVEL_WRITERS_DISABLED
03380 #endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
03381 #if ELPP_COMPILER_MSVC
03382 # define ELPP_VARIADIC_FUNC_MSVC(variadicFunction, variadicArgs) variadicFunction variadicArgs
03383 # define ELPP_VARIADIC_FUNC_MSVC_RUN(variadicFunction, ...) ELPP_VARIADIC_FUNC_MSVC(variadicFunction,
03384     (__VA_ARGS__))
03385 # define el_getVALength(...) ELPP_VARIADIC_FUNC_MSVC_RUN(el_resolveVALength, 0, ## __VA_ARGS__,\
03386     10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
03387 #else
03388 # if ELPP_COMPILER_CLANG
03389 #     define el_getVALength(...) el_resolveVALength(0, __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)

```



```

03390 # else
03391 #     define el_getVALength(...) el_resolveVALength(0, ## __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
03392 0)
03392 # endif // ELPP_COMPILER_CLANG
03393 #endif // ELPP_COMPILER_MSVC
03394 #define el_resolveVALength(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
03395 #define ELPP_WRITE_LOG(writer, level, dispatchAction, ...) \
03396 writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
03397 #define ELPP_WRITE_LOG_IF(writer, condition, level, dispatchAction, ...) if (condition) \
03398 writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
03399 #define ELPP_WRITE_LOG_EVERY_N(writer, occasion, level, dispatchAction, ...) \
03400 ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion) && \
03401 writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
03402 #define ELPP_WRITE_LOG_AFTER_N(writer, n, level, dispatchAction, ...) \
03403 ELPP->validateAfterNCounter(__FILE__, __LINE__, n) && \
03404 writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
03405 #define ELPP_WRITE_LOG_N_TIMES(writer, n, level, dispatchAction, ...) \
03406 ELPP->validateNTimesCounter(__FILE__, __LINE__, n) && \
03407 writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
03408 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
03409 class PerformanceTrackingData {
03410 public:
03411     enum class DataType : base::type::EnumType {
03412         Checkpoint = 1, Complete = 2
03413     };
03414     // Do not use constructor, will run into multiple definition error, use init(PerformanceTracker*)
03415     explicit PerformanceTrackingData(DataType dataType) : m_performanceTracker(nullptr),
03416         m_dataType(dataType), m_firstCheckpoint(false), m_file(""), m_line(0), m_func("") {}
03417     inline const std::string* blockName(void) const;
03418     inline const struct timeval* startTime(void) const;
03419     inline const struct timeval* endTime(void) const;
03420     inline const struct timeval* lastCheckpointTime(void) const;
03421     inline const base::PerformanceTracker* performanceTracker(void) const {
03422         return m_performanceTracker;
03423     }
03424     inline PerformanceTrackingData::DataType dataType(void) const {
03425         return m_dataType;
03426     }
03427     inline bool firstCheckpoint(void) const {
03428         return m_firstCheckpoint;
03429     }
03430     inline std::string checkpointId(void) const {
03431         return m_checkpointId;
03432     }
03433     inline const char* file(void) const {
03434         return m_file;
03435     }
03436     inline base::type::LineNumber line(void) const {
03437         return m_line;
03438     }
03439     inline const char* func(void) const {
03440         return m_func;
03441     }
03442     inline const base::type::string_t* formattedTimeTaken() const {
03443         return &m_formattedTimeTaken;
03444     }
03445     inline const std::string& loggerId(void) const;
03446 private:
03447     base::PerformanceTracker* m_performanceTracker;
03448     base::type::string_t m_formattedTimeTaken;
03449     PerformanceTrackingData::DataType m_dataType;
03450     bool m_firstCheckpoint;
03451     std::string m_checkpointId;
03452     const char* m_file;
03453     base::type::LineNumber m_line;
03454     const char* m_func;
03455     inline void init(base::PerformanceTracker* performanceTracker, bool firstCheckpoint = false) {
03456         m_performanceTracker = performanceTracker;
03457         m_firstCheckpoint = firstCheckpoint;
03458     }
03459     friend class el::base::PerformanceTracker;
03460 };
03461 namespace base {
03462 class PerformanceTracker : public base::threading::ThreadSafe, public Loggable {
03463 public:
03464     PerformanceTracker(const std::string& blockName,
03465         base::TimestampUnit timestampUnit = base::TimestampUnit::Millisecond,
03466         const std::string& loggerId =
03467         std::string(el::base::consts::kPerformanceLoggerId),
03468         bool scopedLog = true, Level level =
03469         base::consts::kPerformanceTrackerDefaultLevel);

```



```

03472     PerformanceTracker(const PerformanceTracker& t) :
03473         m_blockName(t.m_blockName), m_timestampUnit(t.m_timestampUnit), m_loggerId(t.m_loggerId),
03474         m_scopedLog(t.m_scopedLog),
03475         m_level(t.m_level), m_hasChecked(t.m_hasChecked), m_lastCheckpointId(t.m_lastCheckpointId),
03476         m_enabled(t.m_enabled),
03477         m_startTime(t.m_startTime), m_endTime(t.m_endTime), m_lastCheckpointTime(t.m_lastCheckpointTime) {
03478     }
03479     virtual ~PerformanceTracker(void);
03480     void checkpoint(const std::string& id = std::string(), const char* file = __FILE__,
03481                    base::type::LineNumber line = __LINE__,
03482                    const char* func = "");
03483     inline Level level(void) const {
03484         return m_level;
03485     }
03486     private:
03487     std::string m_blockName;
03488     base::TimestampUnit m_timestampUnit;
03489     std::string m_loggerId;
03490     bool m_scopedLog;
03491     Level m_level;
03492     bool m_hasChecked;
03493     std::string m_lastCheckpointId;
03494     bool m_enabled;
03495     struct timeval m_startTime, m_endTime, m_lastCheckpointTime;
03496
03497     PerformanceTracker(void);
03498     friend class el::PerformanceTrackingData;
03499     friend class base::DefaultPerformanceTrackingCallback;
03500
03501     const inline base::type::string_t getFormattedTimeTaken() const {
03502         return getFormattedTimeTaken(m_startTime);
03503     }
03504
03505     const base::type::string_t getFormattedTimeTaken(struct timeval startTime) const;
03506
03507     virtual inline void log(el::base::type::ostream_t& os) const {
03508         os << getFormattedTimeTaken();
03509     }
03510 };
03511 class DefaultPerformanceTrackingCallback : public PerformanceTrackingCallback {
03512     protected:
03513     void handle(const PerformanceTrackingData* data) {
03514         m_data = data;
03515         base::type::stringstream_t ss;
03516         if (m_data->dataType() == PerformanceTrackingData::DataType::Complete) {
03517             ss << ELPP_LITERAL("Executed [") << m_data->blockName()->c_str() << ELPP_LITERAL("] in [") <<
03518                 *m_data->formattedTimeTaken() << ELPP_LITERAL("]");
03519         } else {
03520             ss << ELPP_LITERAL("Performance checkpoint");
03521             if (!m_data->checkpointId().empty()) {
03522                 ss << ELPP_LITERAL(" [") << m_data->checkpointId().c_str() << ELPP_LITERAL("]");
03523             }
03524             ss << ELPP_LITERAL(" for block [") << m_data->blockName()->c_str() << ELPP_LITERAL("] : [") <<
03525                 *m_data->performanceTracker();
03526             if (!ELPP->hasFlag(LoggingFlag::DisablePerformanceTrackingCheckpointComparison)
03527                 && m_data->performanceTracker()->m_hasChecked) {
03528                 ss << ELPP_LITERAL(" ([") << *m_data->formattedTimeTaken() << ELPP_LITERAL("] from ");
03529                 if (m_data->performanceTracker()->m_lastCheckpointId.empty()) {
03530                     ss << ELPP_LITERAL("last checkpoint");
03531                 } else {
03532                     ss << ELPP_LITERAL("checkpoint '") << m_data->performanceTracker()->m_lastCheckpointId.c_str()
03533                         << ELPP_LITERAL("'");
03534                 }
03535             } else {
03536                 ss << ELPP_LITERAL("]");
03537             }
03538         }
03539         el::base::Writer(m_data->performanceTracker()->level(), m_data->file(), m_data->line(),
03540             m_data->func()).construct(1,
03541                 m_data->loggerId().c_str()) << ss.str();
03542     }
03543     private:
03544     const PerformanceTrackingData* m_data;
03545 };
03546 // namespace base
03547 inline const std::string* PerformanceTrackingData::blockName() const {
03548     return const_cast<const std::string*>(&m_performanceTracker->m_blockName);
03549 }
03550 inline const struct timeval* PerformanceTrackingData::startTime() const {
03551     return const_cast<const struct timeval*>(&m_performanceTracker->m_startTime);
03552 }
03553 inline const struct timeval* PerformanceTrackingData::endTime() const {
03554     return const_cast<const struct timeval*>(&m_performanceTracker->m_endTime);
03555 }
03556 inline const struct timeval* PerformanceTrackingData::lastCheckpointTime() const {

```

```

03556     return const_cast<const struct timeval*>(&m_performanceTracker->m_lastCheckpointTime);
03557 }
03558 inline const std::string& PerformanceTrackingData::loggerId(void) const {
03559     return m_performanceTracker->m_loggerId;
03560 }
03561 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
03562 namespace base {
03563 namespace debug {
03564 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
03565 class StackTrace : base::NoCopy {
03566 public:
03567     static const unsigned int kMaxStack = 64;
03568     static const unsigned int kStackStart = 2; // We want to skip c'tor and StackTrace::generateNew()
03569     class StackTraceEntry {
03570     public:
03571         StackTraceEntry(std::size_t index, const std::string& loc, const std::string& demang, const
std::string& hex,
03572             const std::string& addr);
03573         StackTraceEntry(std::size_t index, const std::string& loc) :
03574             m_index(index),
03575             m_location(loc) {
03576         }
03577         std::size_t m_index;
03578         std::string m_location;
03579         std::string m_demangled;
03580         std::string m_hex;
03581         std::string m_addr;
03582         friend std::ostream& operator<<(std::ostream& ss, const StackTraceEntry& si);
03583     private:
03584         StackTraceEntry(void);
03585     };
03586     StackTrace(void) {
03587         generateNew();
03588     }
03589     virtual ~StackTrace(void) {
03590     }
03591     inline std::vector<StackTraceEntry>& getLatestStack(void) {
03592         return m_stack;
03593     }
03594     friend std::ostream& operator<<(std::ostream& os, const StackTrace& st);
03595 private:
03596     std::vector<StackTraceEntry> m_stack;
03597     void generateNew(void);
03598 };
03599 class CrashHandler : base::NoCopy {
03600 public:
03601     typedef void (*Handler)(int);
03602     explicit CrashHandler(bool useDefault);
03603     explicit CrashHandler(const Handler& cHandler) {
03604         setHandler(cHandler);
03605     }
03606     void setHandler(const Handler& cHandler);
03607 private:
03608     Handler m_handler;
03609 };
03610 #else
03611 class CrashHandler {
03612 public:
03613     explicit CrashHandler(bool) {}
03614 };
03615 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
03616 extern base::debug::CrashHandler elCrashHandler;
03617 #define MAKE_LOGGABLE(ClassType, ClassInstance, OutputStreamInstance) \
03618 el::base::type::ostream_t& operator<<(el::base::type::ostream_t& OutputStreamInstance, const ClassType& \
ClassInstance)
03619 class SysLogInitializer {
03620 public:
03621     SysLogInitializer(const char* processIdent, int options = 0, int facility = 0) {
03622 #if defined(ELPP_SYSLOG)
03623         (void)base::consts::kSysLogLoggerId;
03624         openlog(processIdent, options, facility);
03625 #else
03626         ELPP_UNUSED(processIdent);
03627         ELPP_UNUSED(options);
03628         ELPP_UNUSED(facility);
03629 #endif // defined(ELPP_SYSLOG)

```

```

03644     }
03645     virtual ~SysLogInitializer(void) {
03646     #if defined(ELPP_SYSLOG)
03647         closelog();
03648     #endif // defined(ELPP_SYSLOG)
03649     }
03650 };
03651 #define ELPP_INITIALIZE_SYSLOG(id, opt, fac) el::SysLogInitializer elSyslogInit(id, opt, fac)
03653 class Helpers : base::StaticClass {
03654 public:
03655     static inline void setStorage(base::type::StoragePointer storage) {
03656         ELPP = storage;
03657     }
03658 }
03660 static inline base::type::StoragePointer storage() {
03661     return ELPP;
03662 }
03664 static inline void setArgs(int argc, char** argv) {
03665     ELPP->setApplicationArguments(argc, argv);
03666 }
03668 static inline void setArgs(int argc, const char** argv) {
03669     ELPP->setApplicationArguments(argc, const_cast<char**>(argv));
03670 }
03672 static inline void setThreadName(const std::string& name) {
03673     ELPP->setThreadName(name);
03674 }
03675 static inline std::string getThreadName() {
03676     return ELPP->getThreadName(base::threading::getCurrentThreadId());
03677 }
03678 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
03682 static inline void setCrashHandler(const el::base::debug::CrashHandler::Handler& crashHandler) {
03683     el::elCrashHandler.setHandler(crashHandler);
03684 }
03687 static void crashAbort(int sig, const char* sourceFile = "", unsigned int long line = 0);
03693 static void logCrashReason(int sig, bool stackTraceIfAvailable = false,
03694                             Level level = Level::Fatal, const char* logger =
base::consts::kDefaultLoggerId);
03695 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
03698 static inline void installPreRollOutCallback(const PreRollOutCallback& callback) {
03699     ELPP->setPreRollOutCallback(callback);
03700 }
03702 static inline void uninstallPreRollOutCallback(void) {
03703     ELPP->unsetPreRollOutCallback();
03704 }
03706 template <typename T>
03707 static inline bool installLogDispatchCallback(const std::string& id) {
03708     return ELPP->installLogDispatchCallback<T>(id);
03709 }
03711 template <typename T>
03712 static inline void uninstallLogDispatchCallback(const std::string& id) {
03713     ELPP->uninstallLogDispatchCallback<T>(id);
03714 }
03715 template <typename T>
03716 static inline T* logDispatchCallback(const std::string& id) {
03717     return ELPP->logDispatchCallback<T>(id);
03718 }
03719 #if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
03721 template <typename T>
03722 static inline bool installPerformanceTrackingCallback(const std::string& id) {
03723     return ELPP->installPerformanceTrackingCallback<T>(id);
03724 }
03726 template <typename T>
03727 static inline void uninstallPerformanceTrackingCallback(const std::string& id) {
03728     ELPP->uninstallPerformanceTrackingCallback<T>(id);
03729 }
03730 template <typename T>
03731 static inline T* performanceTrackingCallback(const std::string& id) {
03732     return ELPP->performanceTrackingCallback<T>(id);
03733 }
03734 #endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
03736 template <typename T>
03737 static std::string convertTemplateToStdString(const T& templ) {
03738     el::Logger* logger =
03739         ELPP->registeredLoggers()->get(el::base::consts::kDefaultLoggerId);
03740     if (logger == nullptr) {
03741         return std::string();
03742     }
03743     base::MessageBuilder b;
03744     b.initialize(logger);
03745     logger->acquireLock();
03746     b << templ;
03747 #if defined(ELPP_UNICODE)
03748     std::string s = std::string(logger->stream().str().begin(), logger->stream().str().end());
03749 #else
03750     std::string s = logger->stream().str();
03751 #endif // defined(ELPP_UNICODE)
03752     logger->stream().str(ELPP_LITERAL(""));
03753     logger->releaseLock();

```

```

03754     return s;
03755 }
03757 static inline const el::base::utils::CommandLineArgs* commandLineArgs(void) {
03758     return ELPP->commandLineArgs();
03759 }
03762 static inline void reserveCustomFormatSpecifiers(std::size_t size) {
03763     ELPP->m_customFormatSpecifiers.reserve(size);
03764 }
03766 static inline void installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier)
{
03767     ELPP->installCustomFormatSpecifier(customFormatSpecifier);
03768 }
03770 static inline bool uninstallCustomFormatSpecifier(const char* formatSpecifier) {
03771     return ELPP->uninstallCustomFormatSpecifier(formatSpecifier);
03772 }
03774 static inline bool hasCustomFormatSpecifier(const char* formatSpecifier) {
03775     return ELPP->hasCustomFormatSpecifier(formatSpecifier);
03776 }
03777 static inline void validateFileRolling(Logger* logger, Level level) {
03778     if (ELPP == nullptr || logger == nullptr) return;
03779     logger->m_typedConfigurations->validateFileRolling(level, ELPP->preRollOutCallback());
03780 }
03781 };
03783 class Loggers : base::StaticClass {
03784 public:
03786     static Logger* getLogger(const std::string& identity, bool registerIfNotAvailable = true);
03788     static void setDefaultLogBuilder(el::LogBuilderPtr& logBuilderPtr);
03790     template <typename T>
03791     static inline bool installLoggerRegistrationCallback(const std::string& id) {
03792         return ELPP->registeredLoggers()->installLoggerRegistrationCallback<T>(id);
03793     }
03795     template <typename T>
03796     static inline void uninstallLoggerRegistrationCallback(const std::string& id) {
03797         ELPP->registeredLoggers()->uninstallLoggerRegistrationCallback<T>(id);
03798     }
03799     template <typename T>
03800     static inline T* loggerRegistrationCallback(const std::string& id) {
03801         return ELPP->registeredLoggers()->loggerRegistrationCallback<T>(id);
03802     }
03805     static bool unregisterLogger(const std::string& identity);
03807     static bool hasLogger(const std::string& identity);
03809     static Logger* reconfigureLogger(Logger* logger, const Configurations& configurations);
03811     static Logger* reconfigureLogger(const std::string& identity, const Configurations& configurations);
03813     static Logger* reconfigureLogger(const std::string& identity, ConfigurationType configurationType,
03814                                     const std::string& value);
03816     static void reconfigureAllLoggers(const Configurations& configurations);
03818     static inline void reconfigureAllLoggers(ConfigurationType configurationType, const std::string&
value) {
03819         reconfigureAllLoggers(Level::Global, configurationType, value);
03820     }
03822     static void reconfigureAllLoggers(Level level, ConfigurationType configurationType,
03823                                     const std::string& value);
03825     static void setDefaultConfigurations(const Configurations& configurations,
03826                                         bool reconfigureExistingLoggers = false);
03828     static const Configurations* defaultConfigurations(void);
03830     static const base::LogStreamsReferenceMapPtr logStreamsReference(void);
03832     static base::TypedConfigurations defaultTypedConfigurations(void);
03835     static std::vector<std::string>* populateAllLoggerIds(std::vector<std::string>* targetList);
03837     static void configureFromGlobal(const char* globalConfigurationFilePath);
03842     static bool configureFromArg(const char* argKey);
03844     static void flushAll(void);
03846     static inline void addFlag(LoggingFlag flag) {
03847         ELPP->addFlag(flag);
03848     }
03850     static inline void removeFlag(LoggingFlag flag) {
03851         ELPP->removeFlag(flag);
03852     }
03854     static inline bool hasFlag(LoggingFlag flag) {
03855         return ELPP->hasFlag(flag);
03856     }
03858     class ScopedAddFlag {
03859     public:
03860         ScopedAddFlag(LoggingFlag flag) : m_flag(flag) {
03861             Loggers::addFlag(m_flag);
03862         }
03863         ~ScopedAddFlag(void) {
03864             Loggers::removeFlag(m_flag);
03865         }
03866     private:
03867         LoggingFlag m_flag;
03868     };
03870     class ScopedRemoveFlag {
03871     public:
03872         ScopedRemoveFlag(LoggingFlag flag) : m_flag(flag) {
03873             Loggers::removeFlag(m_flag);
03874         }
03875         ~ScopedRemoveFlag(void) {

```

```

03876     Loggers::addFlag(m_flag);
03877 }
03878 private:
03879     LoggingFlag m_flag;
03880 };
03882 static void setLoggingLevel(Level level) {
03883     ELPP->setLoggingLevel(level);
03884 }
03886 static void setVerboseLevel(base::type::VerboseLevel level);
03888 static base::type::VerboseLevel verboseLevel(void);
03890 static void setVModules(const char* modules);
03892 static void clearVModules(void);
03893 };
03894 class VersionInfo : base::StaticClass {
03895 public:
03897     static const std::string version(void);
03898
03900     static const std::string releaseDate(void);
03901 };
03902 } // namespace el
03903 #undef VLOG_IS_ON
03905 #define VLOG_IS_ON(verboseLevel) (ELPP->vRegistry()->allowed(verboseLevel, __FILE__))
03906 #undef TIMED_BLOCK
03907 #undef TIMED_SCOPE
03908 #undef TIMED_SCOPE_IF
03909 #undef TIMED_FUNC
03910 #undef TIMED_FUNC_IF
03911 #undef ELPP_MIN_UNIT
03912 #if defined(ELPP_PERFORMANCE_MICROSECONDS)
03913 #   define ELPP_MIN_UNIT el::base::TimestampUnit::Microsecond
03914 #else
03915 #   define ELPP_MIN_UNIT el::base::TimestampUnit::Millisecond
03916 #endif // (defined(ELPP_PERFORMANCE_MICROSECONDS))
03923 // Note: Do not surround this definition with null macro because of obj instance
03924 #define TIMED_SCOPE_IF(obj, blockname, condition) el::base::type::PerformanceTrackerPtr obj( condition
? \
03925     new el::base::PerformanceTracker(blockname, ELPP_MIN_UNIT) : nullptr )
03926 #define TIMED_SCOPE(obj, blockname) TIMED_SCOPE_IF(obj, blockname, true)
03927 #define TIMED_BLOCK(obj, blockName) for (struct { int i; el::base::type::PerformanceTrackerPtr timer;
} obj = { 0, \
03928     el::base::type::PerformanceTrackerPtr(new el::base::PerformanceTracker(blockName, ELPP_MIN_UNIT)) );
obj.i < 1; ++obj.i)
03935 #define TIMED_FUNC_IF(obj,condition) TIMED_SCOPE_IF(obj, ELPP_FUNC, condition)
03936 #define TIMED_FUNC(obj) TIMED_SCOPE(obj, ELPP_FUNC)
03937 #undef PERFORMANCE_CHECKPOINT
03938 #undef PERFORMANCE_CHECKPOINT_WITH_ID
03939 #define PERFORMANCE_CHECKPOINT(obj) obj->checkpoint(std::string(), __FILE__, __LINE__, ELPP_FUNC)
03940 #define PERFORMANCE_CHECKPOINT_WITH_ID(obj, id) obj->checkpoint(id, __FILE__, __LINE__, ELPP_FUNC)
03941 #undef ELPP_COUNTER
03942 #undef ELPP_COUNTER_POS
03944 #define ELPP_COUNTER (ELPP->hitCounters()->getCounter(__FILE__, __LINE__))
03946 #define ELPP_COUNTER_POS (ELPP_COUNTER == nullptr ? -1 : ELPP_COUNTER->hitCounts())
03947 // Undef levels to support LOG(LEVEL)
03948 #undef INFO
03949 #undef WARNING
03950 #undef DEBUG
03951 #undef ERROR
03952 #undef FATAL
03953 #undef TRACE
03954 #undef VERBOSE
03955 // Undef existing
03956 #undef CINFO
03957 #undef CWARNING
03958 #undef CDEBUG
03959 #undef CFATAL
03960 #undef CERROR
03961 #undef CTRACE
03962 #undef CVERBOSE
03963 #undef CINFO_IF
03964 #undef CWARNING_IF
03965 #undef CDEBUG_IF
03966 #undef CERROR_IF
03967 #undef CFATAL_IF
03968 #undef CTRACE_IF
03969 #undef CVERBOSE_IF
03970 #undef CINFO_EVERY_N
03971 #undef CWARNING_EVERY_N
03972 #undef CDEBUG_EVERY_N
03973 #undef CERROR_EVERY_N
03974 #undef CFATAL_EVERY_N
03975 #undef CTRACE_EVERY_N
03976 #undef CVERBOSE_EVERY_N
03977 #undef CINFO_AFTER_N
03978 #undef CWARNING_AFTER_N
03979 #undef CDEBUG_AFTER_N
03980 #undef CERROR_AFTER_N
03981 #undef CFATAL_AFTER_N

```

```

03982 #undef CTRACE_AFTER_N
03983 #undef CVERBOSE_AFTER_N
03984 #undef CINFO_N_TIMES
03985 #undef CWARNING_N_TIMES
03986 #undef CDEBUG_N_TIMES
03987 #undef CERROR_N_TIMES
03988 #undef CFATAL_N_TIMES
03989 #undef CTRACE_N_TIMES
03990 #undef CVERBOSE_N_TIMES
03991 // Normal logs
03992 #if ELPP_INFO_LOG
03993 # define CINFO(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Info, dispatchAction,
__VA_ARGS__)
03994 #else
03995 # define CINFO(writer, dispatchAction, ...) el::base::NullWriter()
03996 #endif // ELPP_INFO_LOG
03997 #if ELPP_WARNING_LOG
03998 # define CWARNING(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Warning,
dispatchAction, __VA_ARGS__)
03999 #else
04000 # define CWARNING(writer, dispatchAction, ...) el::base::NullWriter()
04001 #endif // ELPP_WARNING_LOG
04002 #if ELPP_DEBUG_LOG
04003 # define CDEBUG(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Debug, dispatchAction,
__VA_ARGS__)
04004 #else
04005 # define CDEBUG(writer, dispatchAction, ...) el::base::NullWriter()
04006 #endif // ELPP_DEBUG_LOG
04007 #if ELPP_ERROR_LOG
04008 # define CERROR(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Error, dispatchAction,
__VA_ARGS__)
04009 #else
04010 # define CERROR(writer, dispatchAction, ...) el::base::NullWriter()
04011 #endif // ELPP_ERROR_LOG
04012 #if ELPP_FATAL_LOG
04013 # define CFATAL(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Fatal, dispatchAction,
__VA_ARGS__)
04014 #else
04015 # define CFATAL(writer, dispatchAction, ...) el::base::NullWriter()
04016 #endif // ELPP_FATAL_LOG
04017 #if ELPP_TRACE_LOG
04018 # define CTRACE(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Trace, dispatchAction,
__VA_ARGS__)
04019 #else
04020 # define CTRACE(writer, dispatchAction, ...) el::base::NullWriter()
04021 #endif // ELPP_TRACE_LOG
04022 #if ELPP_VERBOSE_LOG
04023 # 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__)
04024 #else
04025 # define CVERBOSE(writer, vlevel, dispatchAction, ...) el::base::NullWriter()
04026 #endif // ELPP_VERBOSE_LOG
04027 // Conditional logs
04028 #if ELPP_INFO_LOG
04029 #if ELPP_INFO_LOG
04030 # define CINFO_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Info, dispatchAction, __VA_ARGS__)
04031 #else
04032 # define CINFO_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04033 #endif // ELPP_INFO_LOG
04034 #if ELPP_WARNING_LOG
04035 #if ELPP_WARNING_LOG
04036 # define CWARNING_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Warning, dispatchAction, __VA_ARGS__)
04037 #else
04038 # define CWARNING_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04039 #endif // ELPP_WARNING_LOG
04040 #if ELPP_DEBUG_LOG
04041 #if ELPP_DEBUG_LOG
04042 # define CDEBUG_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Debug, dispatchAction, __VA_ARGS__)
04043 #else
04044 # define CDEBUG_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04045 #endif // ELPP_DEBUG_LOG
04046 #if ELPP_ERROR_LOG
04047 #if ELPP_ERROR_LOG
04048 # define CERROR_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Error, dispatchAction, __VA_ARGS__)
04049 #else
04050 # define CERROR_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04051 #endif // ELPP_ERROR_LOG
04052 #if ELPP_FATAL_LOG
04053 #if ELPP_FATAL_LOG
04054 # define CFATAL_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Fatal, dispatchAction, __VA_ARGS__)
04055 #else
04056 # define CFATAL_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04057 #endif // ELPP_FATAL_LOG
04058 #if ELPP_TRACE_LOG
04059 #if ELPP_TRACE_LOG
04060 # define CTRACE_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Trace, dispatchAction, __VA_ARGS__)
04061 #else

```

```

04062 #else
04063 # define CTRACE_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
04064 #endif // ELPP_TRACE_LOG
04065 #if ELPP_VERBOSE_LOG
04066 # define CVERBOSE_IF(writer, condition_, vlevel, dispatchAction, ...) if (VLOG_IS_ON(vlevel) &&
(condition_)) writer( \
04067 el::Level::Verbose, __FILE__, __LINE__, ELPP_FUNC, dispatchAction,
vlevel).construct(el_getVAlength(__VA_ARGS__), __VA_ARGS__)
04068 #else
04069 # define CVERBOSE_IF(writer, condition_, vlevel, dispatchAction, ...) el::base::NullWriter()
04070 #endif // ELPP_VERBOSE_LOG
04071 // Occasional logs
04072 #if ELPP_INFO_LOG
04073 # define CINFO_EVERY_N(writer, occasion, dispatchAction, ...)\
04074 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Info, dispatchAction, __VA_ARGS__)
04075 #else
04076 # define CINFO_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04077 #endif // ELPP_INFO_LOG
04078 #if ELPP_WARNING_LOG
04079 # define CWARNING_EVERY_N(writer, occasion, dispatchAction, ...)\
04080 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Warning, dispatchAction, __VA_ARGS__)
04081 #else
04082 # define CWARNING_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04083 #endif // ELPP_WARNING_LOG
04084 #if ELPP_DEBUG_LOG
04085 # define CDEBUG_EVERY_N(writer, occasion, dispatchAction, ...)\
04086 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Debug, dispatchAction, __VA_ARGS__)
04087 #else
04088 # define CDEBUG_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04089 #endif // ELPP_DEBUG_LOG
04090 #if ELPP_ERROR_LOG
04091 # define CERROR_EVERY_N(writer, occasion, dispatchAction, ...)\
04092 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Error, dispatchAction, __VA_ARGS__)
04093 #else
04094 # define CERROR_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04095 #endif // ELPP_ERROR_LOG
04096 #if ELPP_FATAL_LOG
04097 # define CFATAL_EVERY_N(writer, occasion, dispatchAction, ...)\
04098 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Fatal, dispatchAction, __VA_ARGS__)
04099 #else
04100 # define CFATAL_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04101 #endif // ELPP_FATAL_LOG
04102 #if ELPP_TRACE_LOG
04103 # define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...)\
04104 ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Trace, dispatchAction, __VA_ARGS__)
04105 #else
04106 # define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
04107 #endif // ELPP_TRACE_LOG
04108 #if ELPP_VERBOSE_LOG
04109 # define CVERBOSE_EVERY_N(writer, occasion, vlevel, dispatchAction, ...)\
04110 CVERBOSE_IF(writer, ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion), vlevel, dispatchAction,
__VA_ARGS__)
04111 #else
04112 # define CVERBOSE_EVERY_N(writer, occasion, vlevel, dispatchAction, ...) el::base::NullWriter()
04113 #endif // ELPP_VERBOSE_LOG
04114 // After N logs
04115 #if ELPP_INFO_LOG
04116 # define CINFO_AFTER_N(writer, n, dispatchAction, ...)\
04117 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
04118 #else
04119 # define CINFO_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04120 #endif // ELPP_INFO_LOG
04121 #if ELPP_WARNING_LOG
04122 # define CWARNING_AFTER_N(writer, n, dispatchAction, ...)\
04123 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
04124 #else
04125 # define CWARNING_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04126 #endif // ELPP_WARNING_LOG
04127 #if ELPP_DEBUG_LOG
04128 # define CDEBUG_AFTER_N(writer, n, dispatchAction, ...)\
04129 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
04130 #else
04131 # define CDEBUG_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04132 #endif // ELPP_DEBUG_LOG
04133 #if ELPP_ERROR_LOG
04134 # define CERROR_AFTER_N(writer, n, dispatchAction, ...)\
04135 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
04136 #else
04137 # define CERROR_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04138 #endif // ELPP_ERROR_LOG
04139 #if ELPP_FATAL_LOG
04140 # define CFATAL_AFTER_N(writer, n, dispatchAction, ...)\
04141 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
04142 #else
04143 # define CFATAL_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04144 #endif // ELPP_FATAL_LOG
04145 #if ELPP_TRACE_LOG

```



```

04146 # define CTRACE_AFTER_N(writer, n, dispatchAction, ...)\
04147 ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)
04148 #else
04149 # define CTRACE_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
04150 #endif // ELPP_TRACE_LOG
04151 #if ELPP_VERBOSE_LOG
04152 # define CVERBOSE_AFTER_N(writer, n, vlevel, dispatchAction, ...)\
04153 CVERBOSE_IF(writer, ELPP->validateAfterNCounter(__FILE__, __LINE__, n), vlevel, dispatchAction,
__VA_ARGS__)
04154 #else
04155 # define CVERBOSE_AFTER_N(writer, n, vlevel, dispatchAction, ...) el::base::NullWriter()
04156 #endif // ELPP_VERBOSE_LOG
04157 // N Times logs
04158 #if ELPP_INFO_LOG
04159 # define CINFO_N_TIMES(writer, n, dispatchAction, ...)\
04160 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
04161 #else
04162 # define CINFO_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04163 #endif // ELPP_INFO_LOG
04164 #if ELPP_WARNING_LOG
04165 # define CWARNING_N_TIMES(writer, n, dispatchAction, ...)\
04166 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
04167 #else
04168 # define CWARNING_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04169 #endif // ELPP_WARNING_LOG
04170 #if ELPP_DEBUG_LOG
04171 # define CDEBUG_N_TIMES(writer, n, dispatchAction, ...)\
04172 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
04173 #else
04174 # define CDEBUG_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04175 #endif // ELPP_DEBUG_LOG
04176 #if ELPP_ERROR_LOG
04177 # define CERROR_N_TIMES(writer, n, dispatchAction, ...)\
04178 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
04179 #else
04180 # define CERROR_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04181 #endif // ELPP_ERROR_LOG
04182 #if ELPP_FATAL_LOG
04183 # define CFATAL_N_TIMES(writer, n, dispatchAction, ...)\
04184 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
04185 #else
04186 # define CFATAL_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04187 #endif // ELPP_FATAL_LOG
04188 #if ELPP_TRACE_LOG
04189 # define CTRACE_N_TIMES(writer, n, dispatchAction, ...)\
04190 ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)
04191 #else
04192 # define CTRACE_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
04193 #endif // ELPP_TRACE_LOG
04194 #if ELPP_VERBOSE_LOG
04195 # define CVERBOSE_N_TIMES(writer, n, vlevel, dispatchAction, ...)\
04196 CVERBOSE_IF(writer, ELPP->validateNTimesCounter(__FILE__, __LINE__, n), vlevel, dispatchAction,
__VA_ARGS__)
04197 #else
04198 # define CVERBOSE_N_TIMES(writer, n, vlevel, dispatchAction, ...) el::base::NullWriter()
04199 #endif // ELPP_VERBOSE_LOG
04200 //
04201 // Custom Loggers - Requires (level, dispatchAction, loggerId/s)
04202 //
04203 // undef existing
04204 #undef CLOG
04205 #undef CLOG_VERBOSE
04206 #undef CVLOG
04207 #undef CLOG_IF
04208 #undef CLOG_VERBOSE_IF
04209 #undef CVLOG_IF
04210 #undef CLOG_EVERY_N
04211 #undef CVLOG_EVERY_N
04212 #undef CLOG_AFTER_N
04213 #undef CVLOG_AFTER_N
04214 #undef CLOG_N_TIMES
04215 #undef CVLOG_N_TIMES
04216 // Normal logs
04217 #define CLOG(LEVEL, ...)\
04218 C##LEVEL(el::base::Writer, el::base::DispatchAction::NormalLog, __VA_ARGS__)
04219 #define CVLOG(vlevel, ...) CVERBOSE(el::base::Writer, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
04220 // Conditional logs
04221 #define CLOG_IF(condition, LEVEL, ...)\
04222 C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::NormalLog, __VA_ARGS__)
04223 #define CVLOG_IF(condition, vlevel, ...)\
04224 CVERBOSE_IF(el::base::Writer, condition, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)
04225 // Hit counts based logs
04226 #define CLOG_EVERY_N(n, LEVEL, ...)\
04227 C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
04228 #define CVLOG_EVERY_N(n, vlevel, ...)\
04229 CVERBOSE_EVERY_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)

```



```

04230 #define CLOG_AFTER_N(n, LEVEL, ...)\
04231 C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04232 #define CVLOG_AFTER_N(n, vlevel, ...)\
04233 CVERBOSE_AFTER_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04234 #define CLOG_N_TIMES(n, LEVEL, ...)\
04235 C##LEVEL##_N_TIMES(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04236 #define CVLOG_N_TIMES(n, vlevel, ...)\
04237 CVERBOSE_N_TIMES(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04238 //
04239 // Default Loggers macro using CLOG(), CLOG_VERBOSE() and CVLOG() macros
04240 //
04241 // undef existing
04242 #undef LOG
04243 #undef VLOG
04244 #undef LOG_IF
04245 #undef VLOG_IF
04246 #undef LOG_EVERY_N
04247 #undef VLOG_EVERY_N
04248 #undef LOG_AFTER_N
04249 #undef VLOG_AFTER_N
04250 #undef LOG_N_TIMES
04251 #undef VLOG_N_TIMES
04252 #undef ELPP_CURR_FILE_LOGGER_ID
04253 #if defined(ELPP_DEFAULT_LOGGER)
04254 # define ELPP_CURR_FILE_LOGGER_ID ELPP_DEFAULT_LOGGER
04255 #else
04256 # define ELPP_CURR_FILE_LOGGER_ID el::base::consts::kDefaultLoggerId
04257 #endif
04258 #undef ELPP_TRACE
04259 #define ELPP_TRACE CLOG(TRACE, ELPP_CURR_FILE_LOGGER_ID)
04260 // Normal logs
04261 #define LOG(LEVEL) CLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04262 #define VLOG(vlevel) CVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
04263 // Conditional logs
04264 #define LOG_IF(condition, LEVEL) CLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04265 #define VLOG_IF(condition, vlevel) CVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04266 // Hit counts based logs
04267 #define LOG_EVERY_N(n, LEVEL) CLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04268 #define VLOG_EVERY_N(n, vlevel) CVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04269 #define LOG_AFTER_N(n, LEVEL) CLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04270 #define VLOG_AFTER_N(n, vlevel) CVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04271 #define LOG_N_TIMES(n, LEVEL) CLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04272 #define VLOG_N_TIMES(n, vlevel) CVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04273 // Generic PLOG()
04274 #undef CPLOG
04275 #undef CPLOG_IF
04276 #undef PLOG
04277 #undef PLOG_IF
04278 #undef DCPLOG
04279 #undef DCPLOG_IF
04280 #undef DPLOG
04281 #undef DPLOG_IF
04282 #define CPLOG(LEVEL, ...)\
04283 C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04284 #define CPLOG_IF(condition, LEVEL, ...)\
04285 C##LEVEL##_IF(el::base::PErrorWriter, condition, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04286 #define DCPLOG(LEVEL, ...)\
04287 if (ELPP_DEBUG_LOG) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04288 #define DCPLOG_IF(condition, LEVEL, ...)\
04289 C##LEVEL##_IF(el::base::PErrorWriter, (ELPP_DEBUG_LOG) && (condition),\
04290 el::base::DispatchAction::NormalLog, __VA_ARGS__)\
04291 #define PLOG(LEVEL) CPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04292 #define PLOG_IF(condition, LEVEL) CPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04293 #define DPLOG(LEVEL) DCPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04294 #define DPLOG_IF(condition, LEVEL) DCPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04295 // Generic SYSLOG()
04296 #undef CSYSLOG
04297 #undef CSYSLOG_IF
04298 #undef CSYSLOG_EVERY_N
04299 #undef CSYSLOG_AFTER_N
04300 #undef SYSLOG
04301 #undef SYSLOG_IF
04302 #undef SYSLOG_EVERY_N
04303 #undef SYSLOG_AFTER_N
04304 #undef SYSLOG_N_TIMES
04305 #undef DCSYSLOG
04306 #undef DCSYSLOG_IF
04307 #undef DCSYSLOG_EVERY_N
04308 #undef DCSYSLOG_AFTER_N
04309 #undef DCSYSLOG_N_TIMES
04310 #undef DSYSLOG
04311 #undef DSYSLOG_IF
04312 #undef DSYSLOG_EVERY_N
04313 #undef DSYSLOG_AFTER_N
04314 #undef DSYSLOG_N_TIMES
04315 #if defined(ELPP_SYSLOG)

```

```

04316 # define CSYSLOG(LEVEL, ...)\
04317 C##LEVEL(el::base::Writer, el::base::DispatchAction::SysLog, __VA_ARGS__)
04318 # define CSYSLOG_IF(condition, LEVEL, ...)\
04319 C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::SysLog, __VA_ARGS__)
04320 # define CSYSLOG_EVERY_N(n, LEVEL, ...) C##LEVEL##_EVERY_N(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
04321 # define CSYSLOG_AFTER_N(n, LEVEL, ...) C##LEVEL##_AFTER_N(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
04322 # define CSYSLOG_N_TIMES(n, LEVEL, ...) C##LEVEL##_N_TIMES(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
04323 # define SYSLOG(LEVEL) CSYSLOG(LEVEL, el::base::consts::kSysLogLoggerId)
04324 # define SYSLOG_IF(condition, LEVEL) CSYSLOG_IF(condition, LEVEL, el::base::consts::kSysLogLoggerId)
04325 # define SYSLOG_EVERY_N(n, LEVEL) CSYSLOG_EVERY_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
04326 # define SYSLOG_AFTER_N(n, LEVEL) CSYSLOG_AFTER_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
04327 # define SYSLOG_N_TIMES(n, LEVEL) CSYSLOG_N_TIMES(n, LEVEL, el::base::consts::kSysLogLoggerId)
04328 # define DCSYSLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) C##LEVEL(el::base::Writer,
el::base::DispatchAction::SysLog, __VA_ARGS__)
04329 # define DCSYSLOG_IF(condition, LEVEL, ...)\
04330 C##LEVEL##_IF(el::base::Writer, (ELPP_DEBUG_LOG) && (condition), el::base::DispatchAction::SysLog,
__VA_ARGS__)
04331 # define DCSYSLOG_EVERY_N(n, LEVEL, ...)\
04332 if (ELPP_DEBUG_LOG) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
04333 # define DCSYSLOG_AFTER_N(n, LEVEL, ...)\
04334 if (ELPP_DEBUG_LOG) C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
04335 # define DCSYSLOG_N_TIMES(n, LEVEL, ...)\
04336 if (ELPP_DEBUG_LOG) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
04337 # define DSYSLOG(LEVEL) DCSYSLOG(LEVEL, el::base::consts::kSysLogLoggerId)
04338 # define DSYSLOG_IF(condition, LEVEL) DCSYSLOG_IF(condition, LEVEL,
el::base::consts::kSysLogLoggerId)
04339 # define DSYSLOG_EVERY_N(n, LEVEL) DCSYSLOG_EVERY_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
04340 # define DSYSLOG_AFTER_N(n, LEVEL) DCSYSLOG_AFTER_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
04341 # define DSYSLOG_N_TIMES(n, LEVEL) DCSYSLOG_N_TIMES(n, LEVEL, el::base::consts::kSysLogLoggerId)
04342 #else
04343 # define CSYSLOG(LEVEL, ...) el::base::NullWriter()
04344 # define CSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
04345 # define CSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
04346 # define CSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
04347 # define CSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
04348 # define SYSLOG(LEVEL) el::base::NullWriter()
04349 # define SYSLOG_IF(condition, LEVEL) el::base::NullWriter()
04350 # define SYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
04351 # define SYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()
04352 # define SYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
04353 # define DCSYSLOG(LEVEL, ...) el::base::NullWriter()
04354 # define DCSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
04355 # define DCSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
04356 # define DCSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
04357 # define DCSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
04358 # define DSYSLOG(LEVEL) el::base::NullWriter()
04359 # define DSYSLOG_IF(condition, LEVEL) el::base::NullWriter()
04360 # define DSYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
04361 # define DSYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()
04362 # define DSYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
04363 #endif // defined(ELPP_SYSLOG)
04364 //
04365 // Custom Debug Only Loggers - Requires (level, loggerId/s)
04366 //
04367 // undef existing
04368 #undef DCLOG
04369 #undef DCVLOG
04370 #undef DCLOG_IF
04371 #undef DCVLOG_IF
04372 #undef DCLOG_EVERY_N
04373 #undef DCVLOG_EVERY_N
04374 #undef DCLOG_AFTER_N
04375 #undef DCVLOG_AFTER_N
04376 #undef DCLOG_N_TIMES
04377 #undef DCVLOG_N_TIMES
04378 // Normal logs
04379 #define DCLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG(LEVEL, __VA_ARGS__)
04380 #define DCLOG_VERBOSE(vlevel, ...) if (ELPP_DEBUG_LOG) CLOG_VERBOSE(vlevel, __VA_ARGS__)
04381 #define DCVLOG(vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG(vlevel, __VA_ARGS__)
04382 // Conditional logs
04383 #define DCLOG_IF(condition, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_IF(condition, LEVEL, __VA_ARGS__)
04384 #define DCVLOG_IF(condition, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_IF(condition, vlevel, __VA_ARGS__)
04385 // Hit counts based logs
04386 #define DCLOG_EVERY_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_EVERY_N(n, LEVEL, __VA_ARGS__)
04387 #define DCVLOG_EVERY_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_EVERY_N(n, vlevel, __VA_ARGS__)
04388 #define DCLOG_AFTER_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_AFTER_N(n, LEVEL, __VA_ARGS__)
04389 #define DCVLOG_AFTER_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_AFTER_N(n, vlevel, __VA_ARGS__)
04390 #define DCLOG_N_TIMES(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_N_TIMES(n, LEVEL, __VA_ARGS__)
04391 #define DCVLOG_N_TIMES(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_N_TIMES(n, vlevel, __VA_ARGS__)
04392 //
04393 // Default Debug Only Loggers macro using CLOG(), CLOG_VERBOSE() and CVLOG() macros

```

```

04394 //
04395 #if !defined(ELPP_NO_DEBUG_MACROS)
04396 // undef existing
04397 #undef DLOG
04398 #undef DVLOG
04399 #undef DLOG_IF
04400 #undef DVLOG_IF
04401 #undef DLOG_EVERY_N
04402 #undef DVLOG_EVERY_N
04403 #undef DLOG_AFTER_N
04404 #undef DVLOG_AFTER_N
04405 #undef DLOG_N_TIMES
04406 #undef DVLOG_N_TIMES
04407 // Normal logs
04408 #define DLOG(LEVEL) DCLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04409 #define DVLOG(vlevel) DCVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
04410 // Conditional logs
04411 #define DLOG_IF(condition, LEVEL) DCLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04412 #define DVLOG_IF(condition, vlevel) DCVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04413 // Hit counts based logs
04414 #define DLOG_EVERY_N(n, LEVEL) DCLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04415 #define DVLOG_EVERY_N(n, vlevel) DCVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04416 #define DLOG_AFTER_N(n, LEVEL) DCLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04417 #define DVLOG_AFTER_N(n, vlevel) DCVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04418 #define DLOG_N_TIMES(n, LEVEL) DCLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
04419 #define DVLOG_N_TIMES(n, vlevel) DCVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
04420 #endif // defined(ELPP_NO_DEBUG_MACROS)
04421 #if !defined(ELPP_NO_CHECK_MACROS)
04422 // Check macros
04423 #undef CCHECK
04424 #undef CPCHECK
04425 #undef CCHECK_EQ
04426 #undef CCHECK_NE
04427 #undef CCHECK_LT
04428 #undef CCHECK_GT
04429 #undef CCHECK_LE
04430 #undef CCHECK_GE
04431 #undef CCHECK_BOUNDS
04432 #undef CCHECK_NOTNULL
04433 #undef CCHECK_STRCASEEQ
04434 #undef CCHECK_STRCASENE
04435 #undef CHECK
04436 #undef PCHECK
04437 #undef CHECK_EQ
04438 #undef CHECK_NE
04439 #undef CHECK_LT
04440 #undef CHECK_GT
04441 #undef CHECK_LE
04442 #undef CHECK_GE
04443 #undef CHECK_BOUNDS
04444 #undef CHECK_NOTNULL
04445 #undef CHECK_STRCASEEQ
04446 #undef CHECK_STRCASENE
04447 #define CCHECK(condition, ...) CLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed: [" <<
04448 #define CPCHECK(condition, ...) CPLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed: [" <<
04449 #define CHECK(condition) CCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
04450 #define PCHECK(condition) CPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
04451 #define CCHECK_EQ(a, b, ...) CCHECK(a == b, __VA_ARGS__)
04452 #define CCHECK_NE(a, b, ...) CCHECK(a != b, __VA_ARGS__)
04453 #define CCHECK_LT(a, b, ...) CCHECK(a < b, __VA_ARGS__)
04454 #define CCHECK_GT(a, b, ...) CCHECK(a > b, __VA_ARGS__)
04455 #define CCHECK_LE(a, b, ...) CCHECK(a <= b, __VA_ARGS__)
04456 #define CCHECK_GE(a, b, ...) CCHECK(a >= b, __VA_ARGS__)
04457 #define CCHECK_BOUNDS(val, min, max, ...) CCHECK(val >= min && val <= max, __VA_ARGS__)
04458 #define CHECK_EQ(a, b) CCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
04459 #define CHECK_NE(a, b) CCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04460 #define CHECK_LT(a, b) CCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
04461 #define CHECK_GT(a, b) CCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
04462 #define CHECK_LE(a, b) CCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04463 #define CHECK_GE(a, b) CCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04464 #define CHECK_BOUNDS(val, min, max) CCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
04465 #define CCHECK_NOTNULL(ptr, ...) CCHECK(ptr != nullptr, __VA_ARGS__)
04466 #define CCHECK_STRREQ(str1, str2, ...) CLOG_IF(!el::base::utils::Str::cStringEq(str1, str2), FATAL,
04467 __VA_ARGS__) \
04468 #define CCHECK_STRNE(str1, str2, ...) CLOG_IF(el::base::utils::Str::cStringEq(str1, str2), FATAL,
04469 __VA_ARGS__) \
04470 #define CCHECK_STRCASEEQ(str1, str2, ...) CLOG_IF(!el::base::utils::Str::cStringCaseEq(str1, str2),
04471 FATAL, __VA_ARGS__) \
04472 #define CCHECK_STRCASENE(str1, str2, ...) CLOG_IF(el::base::utils::Str::cStringCaseEq(str1, str2),
04473 FATAL, __VA_ARGS__) \
04474 #define CHECK_NOTNULL(ptr) CCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)

```

```

04475 #define CHECK_STREQ(str1, str2) CCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04476 #define CHECK_STRNE(str1, str2) CCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04477 #define CHECK_STRCASEEQ(str1, str2) CCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04478 #define CHECK_STRCASENE(str1, str2) CCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04479 #undef DCCHECK
04480 #undef DCCHECK_EQ
04481 #undef DCCHECK_NE
04482 #undef DCCHECK_LT
04483 #undef DCCHECK_GT
04484 #undef DCCHECK_LE
04485 #undef DCCHECK_GE
04486 #undef DCCHECK_BOUNDS
04487 #undef DCCHECK_NOTNULL
04488 #undef DCCHECK_STRCASEEQ
04489 #undef DCCHECK_STRCASENE
04490 #undef DPCHECK
04491 #undef DCHECK
04492 #undef DCHECK_EQ
04493 #undef DCHECK_NE
04494 #undef DCHECK_LT
04495 #undef DCHECK_GT
04496 #undef DCHECK_LE
04497 #undef DCHECK_GE
04498 #undef DCHECK_BOUNDS_
04499 #undef DCHECK_NOTNULL
04500 #undef DCHECK_STRCASEEQ
04501 #undef DCHECK_STRCASENE
04502 #undef DPCHECK
04503 #define DCCHECK(condition, ...) if (ELPP_DEBUG_LOG) CCHECK(condition, __VA_ARGS__)
04504 #define DCCHECK_EQ(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_EQ(a, b, __VA_ARGS__)
04505 #define DCCHECK_NE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_NE(a, b, __VA_ARGS__)
04506 #define DCCHECK_LT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LT(a, b, __VA_ARGS__)
04507 #define DCCHECK_GT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GT(a, b, __VA_ARGS__)
04508 #define DCCHECK_LE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LE(a, b, __VA_ARGS__)
04509 #define DCCHECK_GE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GE(a, b, __VA_ARGS__)
04510 #define DCCHECK_BOUNDS(val, min, max, ...) if (ELPP_DEBUG_LOG) CCHECK_BOUNDS(val, min, max,
__VA_ARGS__)
04511 #define DCCHECK_NOTNULL(ptr, ...) if (ELPP_DEBUG_LOG) CCHECK_NOTNULL((ptr), __VA_ARGS__)
04512 #define DCCHECK_STREQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STREQ(str1, str2, __VA_ARGS__)
04513 #define DCCHECK_STRNE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRNE(str1, str2, __VA_ARGS__)
04514 #define DCCHECK_STRCASEEQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASEEQ(str1, str2,
__VA_ARGS__)
04515 #define DCCHECK_STRCASENE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASENE(str1, str2,
__VA_ARGS__)
04516 #define DPCHECK(condition, ...) if (ELPP_DEBUG_LOG) CPHECK(condition, __VA_ARGS__)
04517 #define DCHECK(condition) DCCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
04518 #define DCHECK_EQ(a, b) DCCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
04519 #define DCHECK_NE(a, b) DCCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04520 #define DCHECK_LT(a, b) DCCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
04521 #define DCHECK_GT(a, b) DCCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
04522 #define DCHECK_LE(a, b) DCCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04523 #define DCHECK_GE(a, b) DCCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
04524 #define DCHECK_BOUNDS(val, min, max) DCCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
04525 #define DCHECK_NOTNULL(ptr) DCCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
04526 #define DCHECK_STREQ(str1, str2) DCCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04527 #define DCHECK_STRNE(str1, str2) DCCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04528 #define DCHECK_STRCASEEQ(str1, str2) DCCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04529 #define DCHECK_STRCASENE(str1, str2) DCCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
04530 #define DPCHECK(condition) DPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
04531 #endif // defined(ELPP_NO_CHECK_MACROS)
04532 #if defined(ELPP_DISABLE_DEFAULT_CRASH_HANDLING)
04533 # define ELPP_USE_DEF_CRASH_HANDLER false
04534 #else
04535 # define ELPP_USE_DEF_CRASH_HANDLER true
04536 #endif // defined(ELPP_DISABLE_DEFAULT_CRASH_HANDLING)
04537 #define ELPP_CRASH_HANDLER_INIT
04538 #define ELPP_INIT_EASYLOGGINGPP(val) \
04539 namespace el { \
04540 namespace base { \
04541 el::base::type::StoragePointer elStorage(val); \
04542 } \
04543 el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER); \
04544 }
04545
04546 #if ELPP_ASYNC_LOGGING
04547 # define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new
el::base::DefaultLogBuilder()),\
04548 new el::base::AsyncDispatchWorker()))
04549 #else
04550 # define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new
el::base::DefaultLogBuilder())))
04551 #endif // ELPP_ASYNC_LOGGING
04552 #define INITIALIZE_NULL_EASYLOGGINGPP \
04553 namespace el { \
04554 namespace base { \
04555 el::base::type::StoragePointer elStorage;\
04556 } \

```

```

04557 el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
04558 }
04559 #define SHARE_EASYLOGGINGPP(initializedStorage)\
04560 namespace el {\
04561 namespace base {\
04562 el::base::type::StoragePointer elStorage(initializedStorage);\
04563 }\
04564 el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
04565 }
04566
04567 #if defined(ELPP_UNICODE)
04568 #   define START_EASYLOGGINGPP(argc, argv) el::Helpers::setArgs(argc, argv);
                                std::locale::global(std::locale(""))
04569 #else
04570 #   define START_EASYLOGGINGPP(argc, argv) el::Helpers::setArgs(argc, argv)
04571 #endif // defined(ELPP_UNICODE)
04572 #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.](#)

```
00001 #include <easylogging++.h>
00002 #include <iostream>
00003
00013 INITIALIZE_EASYLOGGINGPP
00014
00015 namespace WIP {
00016 void exampleEasyLogging();
00017 }
00018
00026 int main(int argc, char *argv[]) {
00027     WIP::exampleEasyLogging();
00028     std::cout << "Hello, World!" << std::endl;
00029     return 0;
00030 }
00031
00039 namespace WIP {
00053 void exampleEasyLogging() {
00054     el::Configurations conf("conf/easylogging.conf");
00055     el::Loggers::reconfigureLogger("default", conf);
00056     el::Loggers::reconfigureAllLoggers(conf);
00057
00058     LOG(INFO) << "My first info log using default logger";
00059 }
00060 } // namespace WIP
```

Index

- ~AbstractRegistry
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [47](#)
- ~CommandLineArgs
 - el::base::utils::CommandLineArgs, [54](#)
- ~Configuration
 - el::Configuration, [58](#)
- ~Configurations
 - el::Configurations, [64](#)
- ~HitCounter
 - el::base::HitCounter, [93](#)
- ~LogBuilder
 - el::LogBuilder, [98](#)
- ~LogFormat
 - el::base::LogFormat, [107](#)
- ~Loggable
 - el::Loggable, [112](#)
- ~Logger
 - el::Logger, [116](#)
- ~NoScopedLock
 - el::base::threading::internal::NoScopedLock< Mutex >, [142](#)
- ~PErrorWriter
 - el::base::PErrorWriter, [152](#)
- ~RegisteredLoggers
 - el::base::RegisteredLoggers, [161](#)
- ~Registry
 - el::base::utils::Registry< T_Ptr, T_Key >, [167](#)
- ~RegistryWithPred
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, [172](#)
- ~ScopedAddFlag
 - el::Loggers::ScopedAddFlag, [174](#)
- ~ScopedRemoveFlag
 - el::Loggers::ScopedRemoveFlag, [175](#)
- ~Storage
 - el::base::Storage, [179](#)
- ~SysLogInitializer
 - el::SysLogInitializer, [196](#)
- ~ThreadSafe
 - el::base::threading::ThreadSafe, [198](#)
- ~TypedConfigurations
 - el::base::TypedConfigurations, [202](#)
- ~Writer
 - el::base::Writer, [216](#)
- abort
 - el::base::utils, [40](#)
- AbstractRegistry
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [47](#)
- acquireLock
 - el::base::threading::ThreadSafe, [198](#)
- addFlag
 - el::base::LogFormat, [108](#)
 - el::base::Storage, [180](#)
 - el::base::utils, [40](#)
 - el::Loggers, [126](#)
- addToBuff
 - el::base::utils::Str, [188](#)
- allowed
 - el::base::VRegistry, [213](#)
- AllowVerboselfModuleNotSpecified
 - el, [18](#)
- And
 - el::base::utils::bitwise, [42](#)
- AppName
 - el::base, [23](#)
- AutoSpacing
 - el, [19](#)
- base::LogDispatcher
 - el::LogDispatchCallback, [101](#)
 - el::LogDispatchData, [103](#)
- base::PerformanceTracker
 - el::PerformanceTrackingCallback, [150](#)
- base::RegisteredLoggers
 - el::LoggerRegistrationCallback, [124](#)
- begin
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [47](#)
- brief
 - el::base::consts, [25](#)
- build
 - el::base::DefaultLogBuilder, [80](#)
 - el::base::TypedConfigurations, [202](#)
 - el::LogBuilder, [99](#)
- buildBaseFilename
 - el::base::utils::File, [83](#)
- buildStrippedFilename
 - el::base::utils::File, [83](#)
- buildTimeInfo
 - el::base::utils::DateTime, [77](#)
- Callback
 - el::Callback< T >, [52](#)
- callback
 - el::base::utils::Utils, [210](#)
- castFromInt

- el::ConfigurationTypeHelper, [72](#)
 - el::LevelHelper, [95](#)
- castToInt
 - el::ConfigurationTypeHelper, [72](#)
 - el::LevelHelper, [95](#)
- cbegin
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [47](#)
- CCHECK
 - easylogging++.h, [271](#)
- CCHECK_BOUNDS
 - easylogging++.h, [271](#)
- CCHECK_EQ
 - easylogging++.h, [271](#)
- CCHECK_GE
 - easylogging++.h, [271](#)
- CCHECK_GT
 - easylogging++.h, [271](#)
- CCHECK_LE
 - easylogging++.h, [272](#)
- CCHECK_LT
 - easylogging++.h, [272](#)
- CCHECK_NE
 - easylogging++.h, [272](#)
- CCHECK_NOTNULL
 - easylogging++.h, [272](#)
- CCHECK_STRCASEEQ
 - easylogging++.h, [272](#)
- CCHECK_STRCASENE
 - easylogging++.h, [273](#)
- CCHECK_STREQ
 - easylogging++.h, [273](#)
- CCHECK_STRNE
 - easylogging++.h, [273](#)
- CDEBUG
 - easylogging++.h, [273](#)
- CDEBUG_AFTER_N
 - easylogging++.h, [274](#)
- CDEBUG EVERY_N
 - easylogging++.h, [274](#)
- CDEBUG_IF
 - easylogging++.h, [274](#)
- CDEBUG_N_TIMES
 - easylogging++.h, [274](#)
- cend
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [48](#)
- CERROR
 - easylogging++.h, [274](#)
- CERROR_AFTER_N
 - easylogging++.h, [275](#)
- CERROR EVERY_N
 - easylogging++.h, [275](#)
- CERROR_IF
 - easylogging++.h, [275](#)
- CERROR_N_TIMES
 - easylogging++.h, [275](#)
- CFATAL
 - easylogging++.h, [275](#)
- CFATAL_AFTER_N
 - easylogging++.h, [276](#)
- CFATAL EVERY_N
 - easylogging++.h, [276](#)
- CFATAL_IF
 - easylogging++.h, [276](#)
- CFATAL_N_TIMES
 - easylogging++.h, [276](#)
- char_t
 - el::base::type, [37](#)
- CHECK
 - easylogging++.h, [276](#)
- CHECK_BOUNDS
 - easylogging++.h, [277](#)
- CHECK_EQ
 - easylogging++.h, [277](#)
- CHECK_GE
 - easylogging++.h, [277](#)
- CHECK_GT
 - easylogging++.h, [277](#)
- CHECK_LE
 - easylogging++.h, [277](#)
- CHECK_LT
 - easylogging++.h, [277](#)
- CHECK_NE
 - easylogging++.h, [278](#)
- CHECK_NOTNULL
 - easylogging++.h, [278](#)
- CHECK_STRCASEEQ
 - easylogging++.h, [278](#)
- CHECK_STRCASENE
 - easylogging++.h, [278](#)
- CHECK_STREQ
 - easylogging++.h, [278](#)
- CHECK_STRNE
 - easylogging++.h, [278](#)
- CINFO
 - easylogging++.h, [279](#)
- CINFO_AFTER_N
 - easylogging++.h, [279](#)
- CINFO EVERY_N
 - easylogging++.h, [279](#)
- CINFO_IF
 - easylogging++.h, [279](#)
- CINFO_N_TIMES
 - easylogging++.h, [279](#)
- clear
 - el::Configurations, [64](#)
- clearBuff
 - el::base::utils::Str, [188](#)
- clearModules
 - el::base::VRegistry, [213](#)
- clearVModules
 - el::Loggers, [126](#)
- CLOG
 - easylogging++.h, [280](#)
- CLOG_AFTER_N

- easylogging++.h, 280
- CLOG_EVERY_N
 - easylogging++.h, 280
- CLOG_IF
 - easylogging++.h, 280
- CLOG_N_TIMES
 - easylogging++.h, 280
- ColoredTerminalOutput
 - el, 18
- CommandLineArgs
 - el::base::utils::CommandLineArgs, 53, 54
- commandLineArgs
 - el::base::Storage, 180
 - el::Helpers, 87
- configString
 - el::ConfigurationStringToTypeItem, 71
- configStringToTypeMap
 - el, 19
- configType
 - el::ConfigurationStringToTypeItem, 71
- Configuration
 - el::Configuration, 58
- configurationFile
 - el::Configurations, 64
- Configurations
 - el::Configurations, 64
- configurations
 - el::base::TypedConfigurations, 202
 - el::Logger, 116
- ConfigurationType
 - el, 17
- configurationType
 - el::Configuration, 58
- configure
 - el::Logger, 116
- configureFromArg
 - el::Loggers, 126
- configureFromGlobal
 - el::Loggers, 126
- const_iterator
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 46
 - el::base::utils::Registry< T_Ptr, T_Key >, 167
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 171
- construct
 - el::base::Writer, 217
- contains
 - el::base::utils::Str, 189
- convertAndAddToBuff
 - el::base::utils::Str, 189
- convertFromString
 - el::ConfigurationTypeHelper, 73
 - el::LevelHelper, 96
- convertTemplateToStdString
 - el::Helpers, 87
- convertToColoredOutput
 - el::LogBuilder, 99
- convertToString
 - el::ConfigurationTypeHelper, 73
 - el::LevelHelper, 96
- CPCHECK
 - easylogging++.h, 281
- CPLOG
 - easylogging++.h, 281
- CPLOG_IF
 - easylogging++.h, 281
- CrashHandler
 - el::base::debug::CrashHandler, 75
- CreateLoggerAutomatically
 - el, 19
- createPath
 - el::base::utils::File, 84
- cStringCaseEq
 - el::base::utils::Str, 189
- cStringEq
 - el::base::utils::Str, 189
- CSYSLOG
 - easylogging++.h, 281
- CSYSLOG_AFTER_N
 - easylogging++.h, 281
- CSYSLOG_EVERY_N
 - easylogging++.h, 282
- CSYSLOG_IF
 - easylogging++.h, 282
- CSYSLOG_N_TIMES
 - easylogging++.h, 282
- CTRACE
 - easylogging++.h, 282
- CTRACE_AFTER_N
 - easylogging++.h, 282
- CTRACE_EVERY_N
 - easylogging++.h, 283
- CTRACE_IF
 - easylogging++.h, 283
- CTRACE_N_TIMES
 - easylogging++.h, 283
- currentHost
 - el::base::utils::OS, 144
- currentUser
 - el::base::utils::OS, 144
- CustomFormatSpecifier
 - el::CustomFormatSpecifier, 76
- customFormatSpecifiers
 - el::base::Storage, 180
- customFormatSpecifiersLock
 - el::base::Storage, 180
- CVERBOSE
 - easylogging++.h, 283
- CVERBOSE_AFTER_N
 - easylogging++.h, 283
- CVERBOSE_EVERY_N
 - easylogging++.h, 284
- CVERBOSE_IF
 - easylogging++.h, 284
- CVERBOSE_N_TIMES

easylogging++.h, [284](#)
 CVLOG
 easylogging++.h, [284](#)
 CVLOG_AFTER_N
 easylogging++.h, [285](#)
 CVLOG_EVERY_N
 easylogging++.h, [285](#)
 CVLOG_IF
 easylogging++.h, [285](#)
 CVLOG_N_TIMES
 easylogging++.h, [285](#)
 CWARNING
 easylogging++.h, [285](#)
 CWARNING_AFTER_N
 easylogging++.h, [286](#)
 CWARNING_EVERY_N
 easylogging++.h, [286](#)
 CWARNING_IF
 easylogging++.h, [286](#)
 CWARNING_N_TIMES
 easylogging++.h, [286](#)

 DateTime
 el::base, [22](#)
 dateTimeFormat
 el::base::LogFormat, [108](#)
 Day
 el::base, [23](#)
 DCCHECK
 easylogging++.h, [286](#)
 DCCHECK_BOUNDS
 easylogging++.h, [287](#)
 DCCHECK_EQ
 easylogging++.h, [287](#)
 DCCHECK_GE
 easylogging++.h, [287](#)
 DCCHECK_GT
 easylogging++.h, [287](#)
 DCCHECK_LE
 easylogging++.h, [287](#)
 DCCHECK_LT
 easylogging++.h, [287](#)
 DCCHECK_NE
 easylogging++.h, [288](#)
 DCCHECK_NOTNULL
 easylogging++.h, [288](#)
 DCCHECK_STRCASEEQ
 easylogging++.h, [288](#)
 DCCHECK_STRCASENE
 easylogging++.h, [288](#)
 DCCHECK_STREQ
 easylogging++.h, [288](#)
 DCCHECK_STRNE
 easylogging++.h, [288](#)
 DCHECK
 easylogging++.h, [289](#)
 DCHECK_BOUNDS
 easylogging++.h, [289](#)
 DCHECK_EQ
 easylogging++.h, [289](#)
 DCHECK_GE
 easylogging++.h, [289](#)
 DCHECK_GT
 easylogging++.h, [289](#)
 DCHECK_LE
 easylogging++.h, [289](#)
 DCHECK_LT
 easylogging++.h, [290](#)
 DCHECK_NE
 easylogging++.h, [290](#)
 DCHECK_NOTNULL
 easylogging++.h, [290](#)
 DCHECK_STRCASEEQ
 easylogging++.h, [290](#)
 DCHECK_STRCASENE
 easylogging++.h, [290](#)
 DCHECK_STREQ
 easylogging++.h, [290](#)
 DCHECK_STRNE
 easylogging++.h, [291](#)
 DCLOG
 easylogging++.h, [291](#)
 DCLOG_AFTER_N
 easylogging++.h, [291](#)
 DCLOG_EVERY_N
 easylogging++.h, [291](#)
 DCLOG_IF
 easylogging++.h, [291](#)
 DCLOG_N_TIMES
 easylogging++.h, [291](#)
 DCLOG_VERBOSE
 easylogging++.h, [292](#)
 DPCHECK
 easylogging++.h, [292](#)
 DCPLOG
 easylogging++.h, [292](#)
 DCPLOG_IF
 easylogging++.h, [292](#)
 DCSYSLOG
 easylogging++.h, [292](#)
 DCSYSLOG_AFTER_N
 easylogging++.h, [292](#)
 DCSYSLOG_EVERY_N
 easylogging++.h, [293](#)
 DCSYSLOG_IF
 easylogging++.h, [293](#)
 DCSYSLOG_N_TIMES
 easylogging++.h, [293](#)
 DCVLOG
 easylogging++.h, [293](#)
 DCVLOG_AFTER_N
 easylogging++.h, [293](#)
 DCVLOG_EVERY_N
 easylogging++.h, [293](#)
 DCVLOG_IF
 easylogging++.h, [294](#)
 DCVLOG_N_TIMES

- easylogging++.h, [294](#)
- Debug
 - el, [18](#)
- deepCopy
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [48](#)
 - el::base::utils::Registry< T_Ptr, T_Key >, [168](#)
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, [172](#)
- defaultConfigurations
 - el::base::RegisteredLoggers, [161](#)
 - el::Loggers, [127](#)
- defaultPreRollOutCallback
 - el::base, [23](#)
- defaultTypedConfigurations
 - el::Loggers, [127](#)
- detail
 - el::base::consts, [25](#)
- DisableApplicationAbortOnFatalLog
 - el, [18](#)
- DisablePerformanceTrackingCheckpointComparison
 - el, [18](#)
- DisableVMModules
 - el, [19](#)
- DisableVMModulesExtensions
 - el, [19](#)
- dispatch
 - el::base::DefaultLogDispatchCallback, [82](#)
 - el::base::LogDispatcher, [105](#)
- DispatchAction
 - el::base, [22](#)
- dispatchAction
 - el::LogDispatchData, [103](#)
- DLOG
 - easylogging++.h, [294](#)
- DLOG_AFTER_N
 - easylogging++.h, [294](#)
- DLOG_EVERY_N
 - easylogging++.h, [294](#)
- DLOG_IF
 - easylogging++.h, [294](#)
- DLOG_N_TIMES
 - easylogging++.h, [295](#)
- DPCHECK
 - easylogging++.h, [295](#)
- DPLOG
 - easylogging++.h, [295](#)
- DPLOG_IF
 - easylogging++.h, [295](#)
- DSYSLOG
 - easylogging++.h, [295](#)
- DSYSLOG_AFTER_N
 - easylogging++.h, [295](#)
- DSYSLOG_EVERY_N
 - easylogging++.h, [296](#)
- DSYSLOG_IF
 - easylogging++.h, [296](#)
- DSYSLOG_N_TIMES
 - easylogging++.h, [296](#)
- DVLOG
 - easylogging++.h, [296](#)
- DVLOG_AFTER_N
 - easylogging++.h, [296](#)
- DVLOG_EVERY_N
 - easylogging++.h, [296](#)
- DVLOG_IF
 - easylogging++.h, [297](#)
- DVLOG_N_TIMES
 - easylogging++.h, [297](#)
- easylogging++.cc
 - ELPP_DEFAULT_LOGGING_FLAGS, [223](#)
- easylogging++.h
 - CCHECK, [271](#)
 - CCHECK_BOUNDS, [271](#)
 - CCHECK_EQ, [271](#)
 - CCHECK_GE, [271](#)
 - CCHECK_GT, [271](#)
 - CCHECK_LE, [272](#)
 - CCHECK_LT, [272](#)
 - CCHECK_NE, [272](#)
 - CCHECK_NOTNULL, [272](#)
 - CCHECK_STRCASEEQ, [272](#)
 - CCHECK_STRCASENE, [273](#)
 - CCHECK_STREQ, [273](#)
 - CCHECK_STRNE, [273](#)
 - CDEBUG, [273](#)
 - CDEBUG_AFTER_N, [274](#)
 - CDEBUG_EVERY_N, [274](#)
 - CDEBUG_IF, [274](#)
 - CDEBUG_N_TIMES, [274](#)
 - CERROR, [274](#)
 - CERROR_AFTER_N, [275](#)
 - CERROR_EVERY_N, [275](#)
 - CERROR_IF, [275](#)
 - CERROR_N_TIMES, [275](#)
 - CFATAL, [275](#)
 - CFATAL_AFTER_N, [276](#)
 - CFATAL_EVERY_N, [276](#)
 - CFATAL_IF, [276](#)
 - CFATAL_N_TIMES, [276](#)
 - CHECK, [276](#)
 - CHECK_BOUNDS, [277](#)
 - CHECK_EQ, [277](#)
 - CHECK_GE, [277](#)
 - CHECK_GT, [277](#)
 - CHECK_LE, [277](#)
 - CHECK_LT, [277](#)
 - CHECK_NE, [278](#)
 - CHECK_NOTNULL, [278](#)
 - CHECK_STRCASEEQ, [278](#)
 - CHECK_STRCASENE, [278](#)
 - CHECK_STREQ, [278](#)
 - CHECK_STRNE, [278](#)
 - CINFO, [279](#)
 - CINFO_AFTER_N, [279](#)
 - CINFO_EVERY_N, [279](#)

CINFO_IF, 279
 CINFO_N_TIMES, 279
 CLOG, 280
 CLOG_AFTER_N, 280
 CLOG_EVERY_N, 280
 CLOG_IF, 280
 CLOG_N_TIMES, 280
 CPCHECK, 281
 CPLOG, 281
 CPLOG_IF, 281
 CSYSLOG, 281
 CSYSLOG_AFTER_N, 281
 CSYSLOG_EVERY_N, 282
 CSYSLOG_IF, 282
 CSYSLOG_N_TIMES, 282
 CTRACE, 282
 CTRACE_AFTER_N, 282
 CTRACE_EVERY_N, 283
 CTRACE_IF, 283
 CTRACE_N_TIMES, 283
 CVERBOSE, 283
 CVERBOSE_AFTER_N, 283
 CVERBOSE_EVERY_N, 284
 CVERBOSE_IF, 284
 CVERBOSE_N_TIMES, 284
 CVLOG, 284
 CVLOG_AFTER_N, 285
 CVLOG_EVERY_N, 285
 CVLOG_IF, 285
 CVLOG_N_TIMES, 285
 CWARNING, 285
 CWARNING_AFTER_N, 286
 CWARNING_EVERY_N, 286
 CWARNING_IF, 286
 CWARNING_N_TIMES, 286
 DCCHECK, 286
 DCCHECK_BOUNDS, 287
 DCCHECK_EQ, 287
 DCCHECK_GE, 287
 DCCHECK_GT, 287
 DCCHECK_LE, 287
 DCCHECK_LT, 287
 DCCHECK_NE, 288
 DCCHECK_NOTNULL, 288
 DCCHECK_STRCASEEQ, 288
 DCCHECK_STRCASENE, 288
 DCCHECK_STREQ, 288
 DCCHECK_STRNE, 288
 DCHECK, 289
 DCHECK_BOUNDS, 289
 DCHECK_EQ, 289
 DCHECK_GE, 289
 DCHECK_GT, 289
 DCHECK_LE, 289
 DCHECK_LT, 290
 DCHECK_NE, 290
 DCHECK_NOTNULL, 290
 DCHECK_STRCASEEQ, 290
 DCHECK_STRCASENE, 290
 DCHECK_STREQ, 290
 DCHECK_STRNE, 291
 DCLOG, 291
 DCLOG_AFTER_N, 291
 DCLOG_EVERY_N, 291
 DCLOG_IF, 291
 DCLOG_N_TIMES, 291
 DCLOG_VERBOSE, 292
 DCPCHECK, 292
 DCPLOG, 292
 DCPLOG_IF, 292
 DCSYSLOG, 292
 DCSYSLOG_AFTER_N, 292
 DCSYSLOG_EVERY_N, 293
 DCSYSLOG_IF, 293
 DCSYSLOG_N_TIMES, 293
 DCVLOG, 293
 DCVLOG_AFTER_N, 293
 DCVLOG_EVERY_N, 293
 DCVLOG_IF, 294
 DCVLOG_N_TIMES, 294
 DLOG, 294
 DLOG_AFTER_N, 294
 DLOG_EVERY_N, 294
 DLOG_IF, 294
 DLOG_N_TIMES, 295
 DPCHECK, 295
 DPLOG, 295
 DPLOG_IF, 295
 DSYSLOG, 295
 DSYSLOG_AFTER_N, 295
 DSYSLOG_EVERY_N, 296
 DSYSLOG_IF, 296
 DSYSLOG_N_TIMES, 296
 DVLOG, 296
 DVLOG_AFTER_N, 296
 DVLOG_EVERY_N, 296
 DVLOG_IF, 297
 DVLOG_N_TIMES, 297
 el_getVAlength, 297
 el_resolveVAlength, 297
 ELPP, 297
 ELPP_ASSERT, 298
 ELPP_ASYNC_LOGGING, 298
 ELPP_COMPILER_CLANG, 298
 ELPP_COMPILER_GCC, 298
 ELPP_COMPILER_INTEL, 298
 ELPP_COMPILER_MSVC, 298
 ELPP_COUNTER, 299
 ELPP_COUNTER_POS, 299
 ELPP_COUT, 299
 ELPP_COUT_LINE, 299
 ELPP_CRASH_HANDLER_INIT, 299
 ELPP_CRT_DBG_WARNINGS, 299
 ELPP_CURR_FILE_LOGGER_ID, 299
 ELPP_CYGWIN, 300
 ELPP_DEBUG_LOG, 300

- ELPP_ERROR_LOG, [300](#)
- ELPP_EXPORT, [300](#)
- ELPP_FATAL_LOG, [300](#)
- ELPP_FINAL, [300](#)
- ELPP_FUNC, [300](#)
- ELPP_INFO_LOG, [300](#)
- ELPP_INIT_EASYLOGGINGPP, [301](#)
- ELPP_INITIALIZE_SYSLOG, [301](#)
- ELPP_INTERNAL_DEBUGGING_ENDL, [301](#)
- ELPP_INTERNAL_DEBUGGING_MSG, [301](#)
- ELPP_INTERNAL_DEBUGGING_OUT_ERROR, [301](#)
- ELPP_INTERNAL_DEBUGGING_OUT_INFO, [301](#)
- ELPP_INTERNAL_DEBUGGING_WRITE_PERROR, [302](#)
- ELPP_INTERNAL_ERROR, [302](#)
- ELPP_INTERNAL_INFO, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG, [303](#)
- ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG, [303](#)
- ELPP_LITERAL, [303](#)
- ELPP_LOGGING_ENABLED, [303](#)
- ELPP_MIN_UNIT, [303](#)
- ELPP_MINGW, [304](#)
- ELPP_OS_AIX, [304](#)
- ELPP_OS_ANDROID, [304](#)
- ELPP_OS_EMSCRIPTEEN, [304](#)
- ELPP_OS_FREEBSD, [304](#)
- ELPP_OS_LINUX, [304](#)
- ELPP_OS_MAC, [304](#)
- ELPP_OS_NETBSD, [304](#)
- ELPP_OS_QNX, [305](#)
- ELPP_OS_SOLARIS, [305](#)
- ELPP_OS_UNIX, [305](#)
- ELPP_OS_WINDOWS, [305](#)
- ELPP_SIMPLE_LOG, [305](#)
- ELPP_STACKTRACE, [305](#)
- ELPP_STRLLEN, [305](#)
- ELPP_THREADING_ENABLED, [306](#)
- ELPP_TRACE, [306](#)
- ELPP_TRACE_LOG, [306](#)
- ELPP_UNUSED, [306](#)
- ELPP_USE_DEF_CRASH_HANDLER, [306](#)
- ELPP_USE_STD_THREADING, [306](#)
- ELPP_VARIADIC_TEMPLATES_SUPPORTED, [306](#)
- ELPP_VERBOSE_LOG, [307](#)
- ELPP_WARNING_LOG, [307](#)
- ELPP_WRITE_LOG, [307](#)
- ELPP_WRITE_LOG_AFTER_N, [307](#)
- ELPP_WRITE_LOG_EVERY_N, [307](#)
- ELPP_WRITE_LOG_IF, [308](#)
- ELPP_WRITE_LOG_N_TIMES, [308](#)
- ELPP_WX_ENABLED, [308](#)
- ELPP_WX_HASH_MAP_ENABLED, [308](#)
- ELPP_WX_PTR_ENABLED, [309](#)
- elptime, [309](#)
- elptime_r, [309](#)
- elptime_s, [309](#)
- INITIALIZE_EASYLOGGINGPP, [309](#)
- INITIALIZE_NULL_EASYLOGGINGPP, [309](#)
- LOG, [309](#)
- LOG_AFTER_N, [310](#)
- LOG_EVERY_N, [310](#)
- LOG_IF, [310](#)
- LOG_N_TIMES, [310](#)
- MAKE_CONTAINERELPP_FRIENDLY, [310](#)
- MAKE_LOGGABLE, [311](#)
- PCHECK, [311](#)
- PERFORMANCE_CHECKPOINT, [311](#)
- PERFORMANCE_CHECKPOINT_WITH_ID, [312](#)
- PLOG, [312](#)
- PLOG_IF, [312](#)
- SHARE_EASYLOGGINGPP, [312](#)
- START_EASYLOGGINGPP, [312](#)
- STRCAT, [312](#)
- STRCPY, [313](#)
- STRERROR, [313](#)
- STRTOK, [313](#)
- SYSLOG, [313](#)
- SYSLOG_AFTER_N, [313](#)
- SYSLOG_EVERY_N, [313](#)
- SYSLOG_IF, [314](#)
- SYSLOG_N_TIMES, [314](#)
- TIMED_BLOCK, [314](#)
- TIMED_FUNC, [314](#)
- TIMED_FUNC_IF, [314](#)
- TIMED_SCOPE, [315](#)
- TIMED_SCOPE_IF, [315](#)
- VLOG, [315](#)
- VLOG_AFTER_N, [315](#)
- VLOG_EVERY_N, [316](#)
- VLOG_IF, [316](#)
- VLOG_IS_ON, [316](#)
- VLOG_N_TIMES, [316](#)
- el, [15](#)
 - AllowVerboseIfModuleNotSpecified, [18](#)
 - AutoSpacing, [19](#)
 - ColoredTerminalOutput, [18](#)
 - configStringToTypeMap, [19](#)
 - ConfigurationType, [17](#)
 - CreateLoggerAutomatically, [19](#)
 - Debug, [18](#)
 - DisableApplicationAbortOnFatalLog, [18](#)
 - DisablePerformanceTrackingCheckpointComparison, [18](#)
 - DisableVModules, [19](#)
 - DisableVModulesExtensions, [19](#)
 - elCrashHandler, [19](#)

- Enabled, [17](#)
- Error, [18](#)
- Fatal, [18](#)
- Filename, [17](#)
- FixedTimeFormat, [19](#)
- Format, [17](#)
- FormatSpecifierValueResolver, [17](#)
- Global, [18](#)
- HierarchicalLogging, [19](#)
- IgnoreSigInt, [19](#)
- ImmediateFlush, [18](#)
- Info, [18](#)
- Level, [18](#)
- LogBuilderPtr, [17](#)
- LogDetailedCrashReason, [18](#)
- LogFlushThreshold, [17](#)
- LoggingFlag, [18](#)
- MaxLogFileSize, [17](#)
- MillisecondsWidth, [17](#)
- MultiLoggerSupport, [18](#)
- NewLineForContainer, [18](#)
- PerformanceTracking, [17](#)
- PreRollOutCallback, [17](#)
- StrictLogFileSizeCheck, [18](#)
- stringToLevelMap, [19](#)
- SubsecondPrecision, [17](#)
- ToFile, [17](#)
- ToStandardOutput, [17](#)
- Trace, [18](#)
- Unknown, [17](#), [18](#)
- Verbose, [18](#)
- Warning, [18](#)
- el::base, [20](#)
 - AppName, [23](#)
 - DateTime, [22](#)
 - Day, [23](#)
 - defaultPreRollOutCallback, [23](#)
 - DispatchAction, [22](#)
 - elStorage, [23](#)
 - File, [22](#)
 - FileBase, [23](#)
 - FileStreamPtr, [21](#)
 - FormatFlags, [22](#)
 - Function, [22](#)
 - Host, [22](#)
 - Hour, [23](#)
 - Level, [23](#)
 - LevelShort, [23](#)
 - Line, [22](#)
 - Location, [22](#)
 - LoggerId, [22](#)
 - LogMessage, [22](#)
 - LogStreamsReferenceMap, [21](#)
 - LogStreamsReferenceMapPtr, [21](#)
 - Microsecond, [23](#)
 - Millisecond, [23](#)
 - MillisecondsWidth, [22](#)
 - Minute, [23](#)
 - None, [22](#)
 - NormalLog, [22](#)
 - Second, [23](#)
 - SysLog, [22](#)
 - ThreadId, [23](#)
 - TimestampUnit, [23](#)
 - User, [22](#)
 - VerboseLevel, [23](#)
- el::base::consts, [23](#)
 - brief, [25](#)
 - detail, [25](#)
 - kAm, [25](#)
 - kAppNameFormatSpecifier, [25](#)
 - kConfigurationComment, [26](#)
 - kConfigurationLevel, [26](#)
 - kConfigurationLoggerId, [26](#)
 - kCrashSignals, [26](#)
 - kCrashSignalsCount, [26](#)
 - kCurrentHostFormatSpecifier, [27](#)
 - kCurrentUserFormatSpecifier, [27](#)
 - kDateTimeFormatSpecifier, [27](#)
 - kDateTimeFormatSpecifierForFilename, [27](#)
 - kDays, [27](#)
 - kDaysAbbrev, [27](#)
 - kDebugLevelLogValue, [27](#)
 - kDebugLevelShortLogValue, [28](#)
 - kDefaultDateTimeFormat, [28](#)
 - kDefaultDateTimeFormatInFilename, [28](#)
 - kDefaultLogFile, [28](#)
 - kDefaultLogFileParam, [28](#)
 - kDefaultLoggerId, [28](#)
 - kDefaultSubsecondPrecision, [28](#)
 - kErrorLevelLogValue, [28](#)
 - kErrorLevelShortLogValue, [29](#)
 - kFatalLevelLogValue, [29](#)
 - kFatalLevelShortLogValue, [29](#)
 - kFilePathSeparator, [29](#)
 - kFormatSpecifierChar, [29](#)
 - kFormatSpecifierCharValue, [29](#)
 - kInfoLevelLogValue, [29](#)
 - kInfoLevelShortLogValue, [29](#)
 - kLogFileBaseFormatSpecifier, [30](#)
 - kLogFileFormatSpecifier, [30](#)
 - kLogFunctionFormatSpecifier, [30](#)
 - kLoggerIdFormatSpecifier, [30](#)
 - kLogLineFormatSpecifier, [30](#)
 - kLogLocationFormatSpecifier, [30](#)
 - kMaxLogPerContainer, [30](#)
 - kMaxLogPerCounter, [31](#)
 - kMaxVerboseLevel, [31](#)
 - kMessageFormatSpecifier, [31](#)
 - kMonths, [31](#)
 - kMonthsAbbrev, [31](#)
 - kNullPointer, [31](#)
 - kPerformanceTrackerDefaultLevel, [31](#)
 - kPm, [32](#)
 - kSeverityLevelFormatSpecifier, [32](#)
 - kSeverityLevelShortFormatSpecifier, [32](#)

- kSourceFilenameMaxLength, 32
- kSourceLineMaxLength, 32
- kThreadIdFormatSpecifier, 32
- kTimeFormats, 32
- kTimeFormatsCount, 33
- kTraceLevelLogValue, 33
- kTraceLevelShortLogValue, 33
- kUnknownHost, 33
- kUnknownUser, 33
- kValidLoggerIdSymbols, 33
- kVerboseLevelFormatSpecifier, 33
- kVerboseLevelLogValue, 34
- kVerboseLevelShortLogValue, 34
- kWarningLevelLogValue, 34
- kWarningLevelShortLogValue, 34
- kYearBase, 34
- name, 34
- numb, 34
- unit, 35
- value, 35
- el::base::debug, 35
- el::base::debug::CrashHandler, 74
 - CrashHandler, 75
- el::base::DefaultLogBuilder, 79
 - build, 80
- el::base::DefaultLogDispatchCallback, 81
 - dispatch, 82
 - el::base::Storage, 184
 - el::base::TypedConfigurations, 207
 - el::LogBuilder, 99
 - el::Logger, 120
 - handle, 82
 - m_data, 82
- el::base::HitCounter, 91
 - ~HitCounter, 93
 - filename, 93
 - HitCounter, 92
 - hitCounts, 93
 - increment, 93
 - lineNumber, 93
 - m_filename, 94
 - m_hitCounts, 94
 - m_lineNumber, 94
 - operator=, 93
 - resetLocation, 94
 - validateHitCounts, 94
- el::base::HitCounter::Predicate, 152
 - m_filename, 153
 - m_lineNumber, 153
 - operator(), 153
 - Predicate, 153
- el::base::LogDispatcher, 104
 - dispatch, 105
 - el::base::Storage, 184
 - el::base::TypedConfigurations, 207
 - el::Logger, 120
 - LogDispatcher, 105
 - m_dispatchAction, 105
 - m_logMessage, 105
 - m_proceed, 105
- el::base::LogFormat, 106
 - ~LogFormat, 107
 - addFlag, 108
 - dateTimeFormat, 108
 - el::Logger, 111
 - flags, 108
 - format, 108
 - hasFlag, 108
 - level, 108
 - log, 109
 - LogFormat, 107
 - m_currentHost, 111
 - m_currentUser, 111
 - m_dateTimeFormat, 111
 - m_flags, 111
 - m_format, 111
 - m_level, 111
 - m_userFormat, 111
 - operator=, 109
 - operator==, 109
 - parseFromFormat, 109
 - updateDateFormat, 110
 - updateFormatSpec, 110
 - userFormat, 110
- el::base::MessageBuilder, 136
 - el::base::Storage, 185
 - el::base::TypedConfigurations, 207
 - el::Logger, 120
 - initialize, 137
 - m_containerLogSeparator, 138
 - m_logger, 138
 - MessageBuilder, 136
 - operator<<, 137
 - writeliterator, 137
- el::base::NoCopy, 138
 - NoCopy, 139
 - operator=, 139
- el::base::NullWriter, 142
 - NullWriter, 143
 - operator bool, 143
 - operator<<, 143
- el::base::PerformanceTracker
 - el::base::Storage, 185
 - el::Logger, 120
- el::base::PErrorWriter, 150
 - ~PErrorWriter, 152
 - el::Logger, 120
 - PErrorWriter, 152
- el::base::RegisteredHitCounters, 155
 - getCounter, 157
 - validateAfterN, 157
 - validateEveryN, 157
 - validateNTimes, 158
- el::base::RegisteredLoggers, 158
 - ~RegisteredLoggers, 161
 - defaultConfigurations, 161

- el::base::Storage, 164
- el::Logger, 120
- flushAll, 161
- get, 162
- has, 162
- installLoggerRegistrationCallback, 162
- loggerRegistrationCallback, 162
- logStreamsReference, 162
- m_defaultConfigurations, 164
- m_defaultLogBuilder, 164
- m_loggerRegistrationCallbacks, 164
- m_logStreamsReference, 164
- RegisteredLoggers, 161
- remove, 162
- setDefaultConfigurations, 163
- setDefaultLogBuilder, 163
- uninstallLoggerRegistrationCallback, 163
- unregister, 163
- unsafeFlushAll, 163
- el::base::StaticClass, 176
 - operator=, 177
 - StaticClass, 177
- el::base::Storage, 177
 - ~Storage, 179
 - addFlag, 180
 - commandLineArgs, 180
 - customFormatSpecifiers, 180
 - customFormatSpecifiersLock, 180
 - el::base::DefaultLogDispatchCallback, 184
 - el::base::LogDispatcher, 184
 - el::base::MessageBuilder, 185
 - el::base::PerformanceTracker, 185
 - el::base::RegisteredLoggers, 164
 - el::base::Writer, 185
 - el::Helpers, 185
 - el::LogBuilder, 185
 - el::Logger, 120
 - flags, 180
 - getThreadName, 180
 - hasCustomFormatSpecifier, 181
 - hasFlag, 181
 - hitCounters, 181
 - installCustomFormatSpecifier, 181
 - installLogDispatchCallback, 181
 - logDispatchCallback, 181
 - m_commandLineArgs, 185
 - m_customFormatSpecifiers, 185
 - m_customFormatSpecifiersLock, 186
 - m_flags, 186
 - m_logDispatchCallbacks, 186
 - m_loggingLevel, 186
 - m_performanceTrackingCallbacks, 186
 - m_preRollOutCallback, 186
 - m_registeredHitCounters, 186
 - m_registeredLoggers, 187
 - m_threadNames, 187
 - m_threadNamesLock, 187
 - m_vRegistry, 187
 - preRollOutCallback, 182
 - registeredLoggers, 182
 - removeFlag, 182
 - setApplicationArguments, 182
 - setFlags, 182
 - setLoggingLevel, 183
 - setPreRollOutCallback, 183
 - setThreadName, 183
 - Storage, 179
 - uninstallCustomFormatSpecifier, 183
 - uninstallLogDispatchCallback, 183
 - unsetPreRollOutCallback, 183
 - validateAfterNCounter, 184
 - validateEveryNCounter, 184
 - validateNTimesCounter, 184
 - vRegistry, 184
- el::base::SubsecondPrecision, 194
 - init, 195
 - m_offset, 195
 - m_width, 195
 - operator==, 195
 - SubsecondPrecision, 194
- el::base::threading, 35
 - getCurrentThreadId, 36
 - Mutex, 36
 - ScopedLock, 36
- el::base::threading::internal, 36
- el::base::threading::internal::NoMutex, 139
 - lock, 140
 - NoMutex, 140
 - try_lock, 140
 - unlock, 140
- el::base::threading::internal::NoScopedLock< Mutex >, 141
 - ~NoScopedLock, 142
 - NoScopedLock, 142
- el::base::threading::ThreadSafe, 197
 - ~ThreadSafe, 198
 - acquireLock, 198
 - lock, 198
 - m_mutex, 198
 - releaseLock, 198
 - ThreadSafe, 198
- el::base::type, 37
 - char_t, 37
 - EnumType, 37
 - fstream_t, 37
 - LineNumber, 37
 - LogDispatchCallbackPtr, 38
 - LoggerRegistrationCallbackPtr, 38
 - ostream_t, 38
 - PerformanceTrackerPtr, 38
 - PerformanceTrackingCallbackPtr, 38
 - StoragePointer, 38
 - string_t, 38
 - stringstream_t, 38
 - VerboseLevel, 39
- el::base::TypedConfigurations, 199

- ~TypedConfigurations, 202
- build, 202
- configurations, 202
- el::base::DefaultLogDispatchCallback, 207
- el::base::LogDispatcher, 207
- el::base::MessageBuilder, 207
- el::base::Writer, 207
- el::Helpers, 207
- enabled, 202
- filename, 203
- fileStream, 203
- getConfigByRef, 203
- getConfigByVal, 203
- getULong, 203
- insertFile, 204
- logFlushThreshold, 204
- logFormat, 204
- m_configurations, 207
- m_enabledMap, 207
- m_filenameMap, 208
- m_fileStreamMap, 208
- m_logFlushThresholdMap, 208
- m_logFormatMap, 208
- m_logStreamsReference, 208
- m_maxLogFileSizeMap, 208
- m_performanceTrackingMap, 208
- m_subsecondPrecisionMap, 209
- m_toFileMap, 209
- m_toStandardOutputMap, 209
- maxLogFileSize, 204
- millisecondsWidth, 204
- performanceTracking, 205
- resolveFilename, 205
- setValue, 205
- subsecondPrecision, 205
- toFile, 205
- toStandardOutput, 206
- TypedConfigurations, 201, 202
- unsafeGetConfigByRef, 206
- unsafeGetConfigByVal, 206
- unsafeValidateFileRolling, 206
- validateFileRolling, 206
- el::base::utils, 39
 - abort, 40
 - addFlag, 40
 - hasFlag, 40
 - operator<<, 40
 - removeFlag, 41
 - safeDelete, 41
- el::base::utils::AbstractRegistry< T_Ptr, Container >, 45
 - ~AbstractRegistry, 47
 - AbstractRegistry, 47
 - begin, 47
 - cbegin, 47
 - cend, 48
 - const_iterator, 46
 - deepCopy, 48
 - empty, 48
 - end, 48
 - iterator, 46
 - list, 49
 - m_list, 50
 - operator!=, 49
 - operator=, 49
 - operator==, 49
 - reinitDeepCopy, 50
 - size, 50
 - unregisterAll, 50
- el::base::utils::bitwise, 41
 - And, 42
 - Not, 42
 - Or, 42
- el::base::utils::CommandLineArgs, 53
 - ~CommandLineArgs, 54
 - CommandLineArgs, 53, 54
 - empty, 54
 - getParamValue, 54
 - hasParam, 54
 - hasParamWithValue, 55
 - m_argc, 56
 - m_argv, 56
 - m_params, 56
 - m_paramsWithValue, 56
 - operator<<, 56
 - setArgs, 55
 - size, 55
- el::base::utils::DateTime, 77
 - buildTimeInfo, 77
 - formatTime, 77
 - getDateTime, 78
 - getTimeDifference, 78
 - gettimeofday, 78
 - parseFormat, 79
 - timevalToString, 79
- el::base::utils::File, 83
 - buildBaseFilename, 83
 - buildStrippedFilename, 83
 - createPath, 84
 - extractPathFromFilename, 84
 - getFileSize, 84
 - newFileStream, 84
 - pathExists, 85
- el::base::utils::OS, 144
 - currentHost, 144
 - currentUser, 144
 - getBashOutput, 145
 - getEnvironmentVariable, 145
 - termSupportsColor, 146
- el::base::utils::Registry< T_Ptr, T_Key >, 165
 - ~Registry, 167
 - const_iterator, 167
 - deepCopy, 168
 - get, 168
 - iterator, 167
 - operator=, 168

- registerNew, 168
- Registry, 167
- unregister, 168
- unregisterAll, 169
- el::base::utils::RegistryWithPred< T_Ptr, Pred >, 169
 - ~RegistryWithPred, 172
 - const_iterator, 171
 - deepCopy, 172
 - get, 172
 - iterator, 171
 - operator<<, 174
 - operator=, 173
 - registerNew, 173
 - RegistryWithPred, 172
 - unregister, 173
 - unregisterAll, 173
- el::base::utils::Str, 187
 - addToBuff, 188
 - clearBuff, 188
 - contains, 189
 - convertAndAddToBuff, 189
 - cStringCaseEq, 189
 - cStringEq, 189
 - endsWith, 189
 - isDigit, 190
 - ltrim, 190
 - replaceAll, 190, 191
 - replaceFirstWithEscape, 191
 - rtrim, 191
 - startsWith, 191
 - toUpper, 192
 - trim, 192
 - wcharPtrToCharPtr, 192
 - wildCardMatch, 192
- el::base::utils::Utils, 209
 - callback, 210
 - installCallback, 210
 - uninstallCallback, 210
- el::base::VRegistry, 211
 - allowed, 213
 - clearModules, 213
 - level, 213
 - m_level, 214
 - m_modules, 214
 - m_pFlags, 215
 - modules, 213
 - setFromArgs, 213
 - setLevel, 214
 - setModules, 214
 - vModulesEnabled, 214
 - VRegistry, 213
- el::base::Writer, 215
 - ~Writer, 216
 - construct, 217
 - el::base::Storage, 185
 - el::base::TypedConfigurations, 207
 - el::Helpers, 218
 - el::Logger, 120
 - initializeLogger, 217
 - m_dispatchAction, 218
 - m_file, 218
 - m_func, 219
 - m_level, 219
 - m_line, 219
 - m_logger, 219
 - m_loggerIds, 219
 - m_messageBuilder, 219
 - m_msg, 219
 - m_proceed, 219
 - m_verboseLevel, 220
 - operator bool, 217
 - operator<<, 217, 218
 - processDispatch, 218
 - triggerDispatch, 218
 - Writer, 216
- el::Callback< T >, 51
 - Callback, 52
 - enabled, 52
 - handle, 52
 - m_enabled, 52
 - setEnabled, 52
- el::Configuration, 57
 - ~Configuration, 58
 - Configuration, 58
 - configurationType, 58
 - level, 58
 - log, 59
 - m_configurationType, 60
 - m_level, 60
 - m_value, 60
 - operator=, 59
 - setValue, 59
 - value, 59
- el::Configuration::Predicate, 153
 - m_configurationType, 154
 - m_level, 154
 - operator(), 154
 - Predicate, 154
- el::Configurations, 60
 - ~Configurations, 64
 - clear, 64
 - configurationFile, 64
 - Configurations, 64
 - el::Loggers, 70
 - get, 65
 - hasConfiguration, 65
 - m_configurationFile, 70
 - m_isFromFile, 70
 - parseFromFile, 66
 - parseFromText, 66
 - set, 67
 - setFromBase, 67
 - setGlobally, 68
 - setRemainingToDefault, 68
 - setDefault, 69
 - unsafeSet, 69

- unsafeSetGlobally, 69
 - unsafeSetIfNotExist, 70
- el::Configurations::Parser, 146
 - el::Loggers, 149
 - ignoreComments, 147
 - isComment, 147
 - isConfig, 147
 - isLevel, 147
 - parseFromFile, 147
 - parseFromText, 148
 - parseLine, 148
- el::ConfigurationStringToTypeItem, 71
 - configString, 71
 - configType, 71
- el::ConfigurationTypeHelper, 72
 - castFromInt, 72
 - castToInt, 72
 - convertFromString, 73
 - convertToString, 73
 - forEachConfigType, 73
 - kMaxValid, 74
 - kMinValid, 74
- el::CustomFormatSpecifier, 75
 - CustomFormatSpecifier, 76
 - formatSpecifier, 76
 - m_formatSpecifier, 76
 - m_resolver, 76
 - operator==, 76
 - resolver, 76
- el::Helpers, 86
 - commandLineArgs, 87
 - convertTemplateToStdString, 87
 - el::base::Storage, 185
 - el::base::TypedConfigurations, 207
 - el::base::Writer, 218
 - el::Logger, 121
 - getThreadName, 87
 - hasCustomFormatSpecifier, 88
 - installCustomFormatSpecifier, 88
 - installLogDispatchCallback, 88
 - installPreRollOutCallback, 88
 - logDispatchCallback, 88
 - reserveCustomFormatSpecifiers, 89
 - setArgs, 89
 - setStorage, 89
 - setThreadName, 90
 - storage, 90
 - uninstallCustomFormatSpecifier, 90
 - uninstallLogDispatchCallback, 90
 - uninstallPreRollOutCallback, 91
 - validateFileRolling, 91
- el::LevelHelper, 95
 - castFromInt, 95
 - castToInt, 95
 - convertFromString, 96
 - convertToString, 96
 - forEachLevel, 96
 - kMaxValid, 97
 - kMinValid, 97
- el::LogBuilder, 97
 - ~LogBuilder, 98
 - build, 99
 - convertToColoredOutput, 99
 - el::base::DefaultLogDispatchCallback, 99
 - el::base::Storage, 185
 - LogBuilder, 98
 - m_termSupportsColor, 99
- el::LogDispatchCallback, 100
 - base::LogDispatcher, 101
 - fileHandle, 101
 - handle, 101
 - m_fileLocks, 101
 - m_fileLocksMapLock, 101
- el::LogDispatchData, 102
 - base::LogDispatcher, 103
 - dispatchAction, 103
 - LogDispatchData, 102
 - logMessage, 103
 - m_dispatchAction, 103
 - m_logMessage, 103
 - setDispatchAction, 103
 - setLogMessage, 103
- el::Loggable, 112
 - ~Loggable, 112
 - log, 113
 - operator<<, 113
- el::Logger, 113
 - ~Logger, 116
 - configurations, 116
 - configure, 116
 - el::base::DefaultLogDispatchCallback, 120
 - el::base::LogDispatcher, 120
 - el::base::LogFormat, 111
 - el::base::MessageBuilder, 120
 - el::base::PerformanceTracker, 120
 - el::base::PErrorWriter, 120
 - el::base::RegisteredLoggers, 120
 - el::base::Storage, 120
 - el::base::Writer, 120
 - el::Helpers, 121
 - el::Loggers, 121
 - el::LogMessage, 121
 - enabled, 116
 - flush, 117
 - id, 117
 - initUnflushedCount, 117
 - isFlushNeeded, 117
 - isValidId, 118
 - log, 118
 - logBuilder, 118
 - Logger, 115, 116
 - m_configurations, 121
 - m_id, 121
 - m_isConfigured, 121
 - m_logBuilder, 121
 - m_logStreamsReference, 122

- m_parentApplicationName, 122
- m_stream, 122
- m_typedConfigurations, 122
- m_unflushedCount, 122
- operator=, 118
- parentApplicationName, 118
- reconfigure, 118
- resolveLoggerFormatSpec, 119
- setLogBuilder, 119
- setParentApplicationName, 119
- stream, 119
- typedConfigurations, 119
- el::LoggerRegistrationCallback, 123
 - base::RegisteredLoggers, 124
- el::Loggers, 124
 - addFlag, 126
 - clearVModules, 126
 - configureFromArg, 126
 - configureFromGlobal, 126
 - defaultConfigurations, 127
 - defaultTypedConfigurations, 127
 - el::Configurations, 70
 - el::Configurations::Parser, 149
 - el::Logger, 121
 - flushAll, 127
 - getLogger, 127
 - hasFlag, 128
 - hasLogger, 128
 - installLoggerRegistrationCallback, 128
 - loggerRegistrationCallback, 128
 - logStreamsReference, 128
 - populateAllLoggerIds, 129
 - reconfigureAllLoggers, 129
 - reconfigureLogger, 130
 - removeFlag, 130
 - setDefaultConfigurations, 131
 - setDefaultLogBuilder, 131
 - setLoggingLevel, 131
 - setVerboseLevel, 131
 - setVModules, 131
 - uninstallLoggerRegistrationCallback, 132
 - unregisterLogger, 132
 - verboseLevel, 132
- el::Loggers::ScopedAddFlag, 174
 - ~ScopedAddFlag, 174
 - m_flag, 175
 - ScopedAddFlag, 174
- el::Loggers::ScopedRemoveFlag, 175
 - ~ScopedRemoveFlag, 175
 - m_flag, 176
 - ScopedRemoveFlag, 175
- el::LogMessage, 133
 - el::Logger, 121
 - file, 134
 - func, 134
 - level, 134
 - line, 134
 - logger, 134
 - LogMessage, 133
 - m_file, 135
 - m_func, 135
 - m_level, 135
 - m_line, 135
 - m_logger, 135
 - m_message, 135
 - m_verboseLevel, 135
 - message, 134
 - verboseLevel, 134
- el::PerformanceTrackingCallback, 149
 - base::PerformanceTracker, 150
- el::StringToLevelItem, 193
 - level, 193
 - levelString, 193
- el::SysLogInitializer, 195
 - ~SysLogInitializer, 196
 - SysLogInitializer, 196
- el::VersionInfo, 210
 - releaseDate, 211
 - version, 211
- el_getVALength
 - easylogging++.h, 297
- el_resolveVALength
 - easylogging++.h, 297
- elCrashHandler
 - el, 19
- ELPP
 - easylogging++.h, 297
- ELPP_ASSERT
 - easylogging++.h, 298
- ELPP_ASYNC_LOGGING
 - easylogging++.h, 298
- ELPP_COMPILER_CLANG
 - easylogging++.h, 298
- ELPP_COMPILER_GCC
 - easylogging++.h, 298
- ELPP_COMPILER_INTEL
 - easylogging++.h, 298
- ELPP_COMPILER_MSVC
 - easylogging++.h, 298
- ELPP_COUNTER
 - easylogging++.h, 299
- ELPP_COUNTER_POS
 - easylogging++.h, 299
- ELPP_COUT
 - easylogging++.h, 299
- ELPP_COUT_LINE
 - easylogging++.h, 299
- ELPP_CRASH_HANDLER_INIT
 - easylogging++.h, 299
- ELPP_CRT_DBG_WARNINGS
 - easylogging++.h, 299
- ELPP_CURR_FILE_LOGGER_ID
 - easylogging++.h, 299
- ELPP_CYGWIN
 - easylogging++.h, 300
- ELPP_DEBUG_LOG

- easylogging++.h, [300](#)
- ELPP_DEFAULT_LOGGING_FLAGS
 - easylogging++.cc, [223](#)
- ELPP_ERROR_LOG
 - easylogging++.h, [300](#)
- ELPP_EXPORT
 - easylogging++.h, [300](#)
- ELPP_FATAL_LOG
 - easylogging++.h, [300](#)
- ELPP_FINAL
 - easylogging++.h, [300](#)
- ELPP_FUNC
 - easylogging++.h, [300](#)
- ELPP_INFO_LOG
 - easylogging++.h, [300](#)
- ELPP_INIT_EASYLOGGINGPP
 - easylogging++.h, [301](#)
- ELPP_INITIALIZE_SYSLOG
 - easylogging++.h, [301](#)
- ELPP_INTERNAL_DEBUGGING_ENDL
 - easylogging++.h, [301](#)
- ELPP_INTERNAL_DEBUGGING_MSG
 - easylogging++.h, [301](#)
- ELPP_INTERNAL_DEBUGGING_OUT_ERROR
 - easylogging++.h, [301](#)
- ELPP_INTERNAL_DEBUGGING_OUT_INFO
 - easylogging++.h, [301](#)
- ELPP_INTERNAL_DEBUGGING_WRITE_PERROR
 - easylogging++.h, [302](#)
- ELPP_INTERNAL_ERROR
 - easylogging++.h, [302](#)
- ELPP_INTERNAL_INFO
 - easylogging++.h, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG
 - easylogging++.h, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG
 - easylogging++.h, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG
 - easylogging++.h, [302](#)
- ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG
 - easylogging++.h, [303](#)
- ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG
 - easylogging++.h, [303](#)
- ELPP_LITERAL
 - easylogging++.h, [303](#)
- ELPP_LOGGING_ENABLED
 - easylogging++.h, [303](#)
- ELPP_MIN_UNIT
 - easylogging++.h, [303](#)
- ELPP_MINGW
 - easylogging++.h, [304](#)
- ELPP_OS_AIX
 - easylogging++.h, [304](#)
- ELPP_OS_ANDROID
 - easylogging++.h, [304](#)
- ELPP_OS_EMSCRIPTEN
 - easylogging++.h, [304](#)
- ELPP_OS_FREEBSD
 - easylogging++.h, [304](#)
- ELPP_OS_LINUX
 - easylogging++.h, [304](#)
- ELPP_OS_MAC
 - easylogging++.h, [304](#)
- ELPP_OS_NETBSD
 - easylogging++.h, [304](#)
- ELPP_OS_QNX
 - easylogging++.h, [305](#)
- ELPP_OS_SOLARIS
 - easylogging++.h, [305](#)
- ELPP_OS_UNIX
 - easylogging++.h, [305](#)
- ELPP_OS_WINDOWS
 - easylogging++.h, [305](#)
- ELPP_SIMPLE_LOG
 - easylogging++.h, [305](#)
- ELPP_STACKTRACE
 - easylogging++.h, [305](#)
- ELPP_STRLLEN
 - easylogging++.h, [305](#)
- ELPP_THREADING_ENABLED
 - easylogging++.h, [306](#)
- ELPP_TRACE
 - easylogging++.h, [306](#)
- ELPP_TRACE_LOG
 - easylogging++.h, [306](#)
- ELPP_UNUSED
 - easylogging++.h, [306](#)
- ELPP_USE_DEF_CRASH_HANDLER
 - easylogging++.h, [306](#)
- ELPP_USE_STD_THREADING
 - easylogging++.h, [306](#)
- ELPP_VARIADIC_TEMPLATES_SUPPORTED
 - easylogging++.h, [306](#)
- ELPP_VERBOSE_LOG
 - easylogging++.h, [307](#)
- ELPP_WARNING_LOG
 - easylogging++.h, [307](#)
- ELPP_WRITE_LOG
 - easylogging++.h, [307](#)
- ELPP_WRITE_LOG_AFTER_N
 - easylogging++.h, [307](#)
- ELPP_WRITE_LOG_EVERY_N
 - easylogging++.h, [307](#)
- ELPP_WRITE_LOG_IF
 - easylogging++.h, [308](#)
- ELPP_WRITE_LOG_N_TIMES
 - easylogging++.h, [308](#)
- ELPP_WX_ENABLED
 - easylogging++.h, [308](#)
- ELPP_WX_HASH_MAP_ENABLED
 - easylogging++.h, [308](#)
- ELPP_WX_PTR_ENABLED
 - easylogging++.h, [309](#)
- elpptime
 - easylogging++.h, [309](#)
- elpptime_r

- easylogging++.h, 309
- elpptime_s
 - easylogging++.h, 309
- elStorage
 - el::base, 23
- empty
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 48
 - el::base::utils::CommandLineArgs, 54
- Enabled
 - el, 17
- enabled
 - el::base::TypedConfigurations, 202
 - el::Callback< T >, 52
 - el::Logger, 116
- end
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 48
- endsWith
 - el::base::utils::Str, 189
- EnumType
 - el::base::type, 37
- Error
 - el, 18
- exampleEasyLogging
 - WIP, 43
- extractPathFromFilename
 - el::base::utils::File, 84
- Fatal
 - el, 18
- File
 - el::base, 22
- file
 - el::LogMessage, 134
- FileBase
 - el::base, 23
- fileHandle
 - el::LogDispatchCallback, 101
- Filename
 - el, 17
- filename
 - el::base::HitCounter, 93
 - el::base::TypedConfigurations, 203
- fileStream
 - el::base::TypedConfigurations, 203
- FileStreamPtr
 - el::base, 21
- FixedTimeFormat
 - el, 19
- flags
 - el::base::LogFormat, 108
 - el::base::Storage, 180
- flush
 - el::Logger, 117
- flushAll
 - el::base::RegisteredLoggers, 161
 - el::Loggers, 127
- forEachConfigType
 - el::ConfigurationTypeHelper, 73
- forEachLevel
 - el::LevelHelper, 96
- Format
 - el, 17
- format
 - el::base::LogFormat, 108
- FormatFlags
 - el::base, 22
- formatSpecifier
 - el::CustomFormatSpecifier, 76
- FormatSpecifierValueResolver
 - el, 17
- formatTime
 - el::base::utils::DateTime, 77
- fstream_t
 - el::base::type, 37
- func
 - el::LogMessage, 134
- Function
 - el::base, 22
- get
 - el::base::RegisteredLoggers, 162
 - el::base::utils::Registry< T_Ptr, T_Key >, 168
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 172
 - el::Configurations, 65
- getBashOutput
 - el::base::utils::OS, 145
- getConfigByRef
 - el::base::TypedConfigurations, 203
- getConfigByVal
 - el::base::TypedConfigurations, 203
- getCounter
 - el::base::RegisteredHitCounters, 157
- getCurrentThreadId
 - el::base::threading, 36
- getDateTime
 - el::base::utils::DateTime, 78
- getEnvironmentVariable
 - el::base::utils::OS, 145
- getLogger
 - el::Loggers, 127
- getParamValue
 - el::base::utils::CommandLineArgs, 54
- getSizeOfFile
 - el::base::utils::File, 84
- getThreadName
 - el::base::Storage, 180
 - el::Helpers, 87
- getTimeDifference
 - el::base::utils::DateTime, 78
- gettimeofday
 - el::base::utils::DateTime, 78
- getULong
 - el::base::TypedConfigurations, 203
- Global
 - el, 18

- handle
 - el::base::DefaultLogDispatchCallback, [82](#)
 - el::Callback< T >, [52](#)
 - el::LogDispatchCallback, [101](#)
- has
 - el::base::RegisteredLoggers, [162](#)
- hasConfiguration
 - el::Configurations, [65](#)
- hasCustomFormatSpecifier
 - el::base::Storage, [181](#)
 - el::Helpers, [88](#)
- hasFlag
 - el::base::LogFormat, [108](#)
 - el::base::Storage, [181](#)
 - el::base::utils, [40](#)
 - el::Loggers, [128](#)
- hasLogger
 - el::Loggers, [128](#)
- hasParam
 - el::base::utils::CommandLineArgs, [54](#)
- hasParamWithValue
 - el::base::utils::CommandLineArgs, [55](#)
- HierarchicalLogging
 - el, [19](#)
- HitCounter
 - el::base::HitCounter, [92](#)
- hitCounters
 - el::base::Storage, [181](#)
- hitCounts
 - el::base::HitCounter, [93](#)
- Host
 - el::base, [22](#)
- Hour
 - el::base, [23](#)
- id
 - el::Logger, [117](#)
- ignoreComments
 - el::Configurations::Parser, [147](#)
- IgnoreSigInt
 - el, [19](#)
- ImmediateFlush
 - el, [18](#)
- increment
 - el::base::HitCounter, [93](#)
- Info
 - el, [18](#)
- init
 - el::base::SubsecondPrecision, [195](#)
- initialize
 - el::base::MessageBuilder, [137](#)
- INITIALIZE_EASYLOGGINGPP
 - easylogging++.h, [309](#)
- INITIALIZE_NULL_EASYLOGGINGPP
 - easylogging++.h, [309](#)
- initializeLogger
 - el::base::Writer, [217](#)
- initUnflushedCount
 - el::Logger, [117](#)
- insertFile
 - el::base::TypedConfigurations, [204](#)
- installCallback
 - el::base::utils::Utils, [210](#)
- installCustomFormatSpecifier
 - el::base::Storage, [181](#)
 - el::Helpers, [88](#)
- installLogDispatchCallback
 - el::base::Storage, [181](#)
 - el::Helpers, [88](#)
- installLoggerRegistrationCallback
 - el::base::RegisteredLoggers, [162](#)
 - el::Loggers, [128](#)
- installPreRollOutCallback
 - el::Helpers, [88](#)
- isComment
 - el::Configurations::Parser, [147](#)
- isConfig
 - el::Configurations::Parser, [147](#)
- isDigit
 - el::base::utils::Str, [190](#)
- isFlushNeeded
 - el::Logger, [117](#)
- isLevel
 - el::Configurations::Parser, [147](#)
- isValidId
 - el::Logger, [118](#)
- iterator
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [46](#)
 - el::base::utils::Registry< T_Ptr, T_Key >, [167](#)
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, [171](#)
- kAm
 - el::base::consts, [25](#)
- kAppNameFormatSpecifier
 - el::base::consts, [25](#)
- kConfigurationComment
 - el::base::consts, [26](#)
- kConfigurationLevel
 - el::base::consts, [26](#)
- kConfigurationLoggerId
 - el::base::consts, [26](#)
- kCrashSignals
 - el::base::consts, [26](#)
- kCrashSignalsCount
 - el::base::consts, [26](#)
- kCurrentHostFormatSpecifier
 - el::base::consts, [27](#)
- kCurrentUserFormatSpecifier
 - el::base::consts, [27](#)
- kDateTimeFormatSpecifier
 - el::base::consts, [27](#)
- kDateTimeFormatSpecifierForFilename
 - el::base::consts, [27](#)
- kDays
 - el::base::consts, [27](#)
- kDaysAbbrev

- el::base::consts, 27
- kDebugLevelLogValue
 - el::base::consts, 27
- kDebugLevelShortLogValue
 - el::base::consts, 28
- kDefaultDateTimeFormat
 - el::base::consts, 28
- kDefaultDateTimeFormatInFilename
 - el::base::consts, 28
- kDefaultLogFile
 - el::base::consts, 28
- kDefaultLogFileParam
 - el::base::consts, 28
- kDefaultLoggerId
 - el::base::consts, 28
- kDefaultSubsecondPrecision
 - el::base::consts, 28
- kErrorLevelLogValue
 - el::base::consts, 28
- kErrorLevelShortLogValue
 - el::base::consts, 29
- kFatalLevelLogValue
 - el::base::consts, 29
- kFatalLevelShortLogValue
 - el::base::consts, 29
- kFilePathSeparator
 - el::base::consts, 29
- kFormatSpecifierChar
 - el::base::consts, 29
- kFormatSpecifierCharValue
 - el::base::consts, 29
- kInfoLevelLogValue
 - el::base::consts, 29
- kInfoLevelShortLogValue
 - el::base::consts, 29
- kLogFileBaseFormatSpecifier
 - el::base::consts, 30
- kLogFileFormatSpecifier
 - el::base::consts, 30
- kLogFunctionFormatSpecifier
 - el::base::consts, 30
- kLoggerIdFormatSpecifier
 - el::base::consts, 30
- kLogLineFormatSpecifier
 - el::base::consts, 30
- kLogLocationFormatSpecifier
 - el::base::consts, 30
- kMaxLogPerContainer
 - el::base::consts, 30
- kMaxLogPerCounter
 - el::base::consts, 31
- kMaxValid
 - el::ConfigurationTypeHelper, 74
 - el::LevelHelper, 97
- kMaxVerboseLevel
 - el::base::consts, 31
- kMessageFormatSpecifier
 - el::base::consts, 31
- kMinValid
 - el::ConfigurationTypeHelper, 74
 - el::LevelHelper, 97
- kMonths
 - el::base::consts, 31
- kMonthsAbbrev
 - el::base::consts, 31
- kNullPointer
 - el::base::consts, 31
- kPerformanceTrackerDefaultLevel
 - el::base::consts, 31
- kPm
 - el::base::consts, 32
- kSeverityLevelFormatSpecifier
 - el::base::consts, 32
- kSeverityLevelShortFormatSpecifier
 - el::base::consts, 32
- kSourceFilenameMaxLength
 - el::base::consts, 32
- kSourceLineMaxLength
 - el::base::consts, 32
- kThreadIdFormatSpecifier
 - el::base::consts, 32
- kTimeFormats
 - el::base::consts, 32
- kTimeFormatsCount
 - el::base::consts, 33
- kTraceLevelLogValue
 - el::base::consts, 33
- kTraceLevelShortLogValue
 - el::base::consts, 33
- kUnknownHost
 - el::base::consts, 33
- kUnknownUser
 - el::base::consts, 33
- kValidLoggerIdSymbols
 - el::base::consts, 33
- kVerboseLevelFormatSpecifier
 - el::base::consts, 33
- kVerboseLevelLogValue
 - el::base::consts, 34
- kVerboseLevelShortLogValue
 - el::base::consts, 34
- kWarningLevelLogValue
 - el::base::consts, 34
- kWarningLevelShortLogValue
 - el::base::consts, 34
- kYearBase
 - el::base::consts, 34
- Level
 - el, 18
 - el::base, 23
- level
 - el::base::LogFormat, 108
 - el::base::VRegistry, 213
 - el::Configuration, 58
 - el::LogMessage, 134
 - el::StringToLevelItem, 193

- LevelShort
 - el::base, [23](#)
- levelString
 - el::StringToLevelItem, [193](#)
- lib/easylogging++.cc, [221](#), [223](#)
- lib/easylogging++.h, [260](#), [317](#)
- Line
 - el::base, [22](#)
- line
 - el::LogMessage, [134](#)
- LineNumber
 - el::base::type, [37](#)
- lineNumber
 - el::base::HitCounter, [93](#)
- list
 - el::base::utils::AbstractRegistry< T_Ptr, Container
>, [49](#)
- Location
 - el::base, [22](#)
- lock
 - el::base::threading::internal::NoMutex, [140](#)
 - el::base::threading::ThreadSafe, [198](#)
- LOG
 - easylogging++.h, [309](#)
- log
 - el::base::LogFormat, [109](#)
 - el::Configuration, [59](#)
 - el::Loggable, [113](#)
 - el::Logger, [118](#)
- LOG_AFTER_N
 - easylogging++.h, [310](#)
- LOG_EVERY_N
 - easylogging++.h, [310](#)
- LOG_IF
 - easylogging++.h, [310](#)
- LOG_N_TIMES
 - easylogging++.h, [310](#)
- LogBuilder
 - el::LogBuilder, [98](#)
- logBuilder
 - el::Logger, [118](#)
- LogBuilderPtr
 - el, [17](#)
- LogDetailedCrashReason
 - el, [18](#)
- logDispatchCallback
 - el::base::Storage, [181](#)
 - el::Helpers, [88](#)
- LogDispatchCallbackPtr
 - el::base::type, [38](#)
- LogDispatchData
 - el::LogDispatchData, [102](#)
- LogDispatcher
 - el::base::LogDispatcher, [105](#)
- LogFlushThreshold
 - el, [17](#)
- logFlushThreshold
 - el::base::TypedConfigurations, [204](#)
- LogFormat
 - el::base::LogFormat, [107](#)
- logFormat
 - el::base::TypedConfigurations, [204](#)
- Logger
 - el::Logger, [115](#), [116](#)
- logger
 - el::LogMessage, [134](#)
- LoggerId
 - el::base, [22](#)
- loggerRegistrationCallback
 - el::base::RegisteredLoggers, [162](#)
 - el::Loggers, [128](#)
- LoggerRegistrationCallbackPtr
 - el::base::type, [38](#)
- LoggingFlag
 - el, [18](#)
- LogMessage
 - el::base, [22](#)
 - el::LogMessage, [133](#)
- logMessage
 - el::LogDispatchData, [103](#)
- logStreamsReference
 - el::base::RegisteredLoggers, [162](#)
 - el::Loggers, [128](#)
- LogStreamsReferenceMap
 - el::base, [21](#)
- LogStreamsReferenceMapPtr
 - el::base, [21](#)
- ltrim
 - el::base::utils::Str, [190](#)
- m_argc
 - el::base::utils::CommandLineArgs, [56](#)
- m_argv
 - el::base::utils::CommandLineArgs, [56](#)
- m_commandLineArgs
 - el::base::Storage, [185](#)
- m_configurationFile
 - el::Configurations, [70](#)
- m_configurations
 - el::base::TypedConfigurations, [207](#)
 - el::Logger, [121](#)
- m_configurationType
 - el::Configuration, [60](#)
 - el::Configuration::Predicate, [154](#)
- m_containerLogSeparator
 - el::base::MessageBuilder, [138](#)
- m_currentHost
 - el::base::LogFormat, [111](#)
- m_currentUser
 - el::base::LogFormat, [111](#)
- m_customFormatSpecifiers
 - el::base::Storage, [185](#)
- m_customFormatSpecifiersLock
 - el::base::Storage, [186](#)
- m_data
 - el::base::DefaultLogDispatchCallback, [82](#)
- m_dateTimeFormat

- el::base::LogFormat, 111
- m_defaultConfigurations
 - el::base::RegisteredLoggers, 164
- m_defaultLogBuilder
 - el::base::RegisteredLoggers, 164
- m_dispatchAction
 - el::base::LogDispatcher, 105
 - el::base::Writer, 218
 - el::LogDispatchData, 103
- m_enabled
 - el::Callback< T >, 52
- m_enabledMap
 - el::base::TypedConfigurations, 207
- m_file
 - el::base::Writer, 218
 - el::LogMessage, 135
- m_fileLocks
 - el::LogDispatchCallback, 101
- m_fileLocksMapLock
 - el::LogDispatchCallback, 101
- m_filename
 - el::base::HitCounter, 94
 - el::base::HitCounter::Predicate, 153
- m_filenameMap
 - el::base::TypedConfigurations, 208
- m_fileStreamMap
 - el::base::TypedConfigurations, 208
- m_flag
 - el::Loggers::ScopedAddFlag, 175
 - el::Loggers::ScopedRemoveFlag, 176
- m_flags
 - el::base::LogFormat, 111
 - el::base::Storage, 186
- m_format
 - el::base::LogFormat, 111
- m_formatSpecifier
 - el::CustomFormatSpecifier, 76
- m_func
 - el::base::Writer, 219
 - el::LogMessage, 135
- m_hitCounts
 - el::base::HitCounter, 94
- m_id
 - el::Logger, 121
- m_isConfigured
 - el::Logger, 121
- m_isFromFile
 - el::Configurations, 70
- m_level
 - el::base::LogFormat, 111
 - el::base::VRegistry, 214
 - el::base::Writer, 219
 - el::Configuration, 60
 - el::Configuration::Predicate, 154
 - el::LogMessage, 135
- m_line
 - el::base::Writer, 219
 - el::LogMessage, 135
- m_lineNumber
 - el::base::HitCounter, 94
 - el::base::HitCounter::Predicate, 153
- m_list
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 50
- m_logBuilder
 - el::Logger, 121
- m_logDispatchCallbacks
 - el::base::Storage, 186
- m_logFlushThresholdMap
 - el::base::TypedConfigurations, 208
- m_logFormatMap
 - el::base::TypedConfigurations, 208
- m_logger
 - el::base::MessageBuilder, 138
 - el::base::Writer, 219
 - el::LogMessage, 135
- m_loggerIds
 - el::base::Writer, 219
- m_loggerRegistrationCallbacks
 - el::base::RegisteredLoggers, 164
- m_loggingLevel
 - el::base::Storage, 186
- m_logMessage
 - el::base::LogDispatcher, 105
 - el::LogDispatchData, 103
- m_logStreamsReference
 - el::base::RegisteredLoggers, 164
 - el::base::TypedConfigurations, 208
 - el::Logger, 122
- m_maxLogFileSizeMap
 - el::base::TypedConfigurations, 208
- m_message
 - el::LogMessage, 135
- m_messageBuilder
 - el::base::Writer, 219
- m_modules
 - el::base::VRegistry, 214
- m_msg
 - el::base::Writer, 219
- m_mutex
 - el::base::threading::ThreadSafe, 198
- m_offset
 - el::base::SubsecondPrecision, 195
- m_params
 - el::base::utils::CommandLineArgs, 56
- m_paramsWithValue
 - el::base::utils::CommandLineArgs, 56
- m_parentApplicationName
 - el::Logger, 122
- m_performanceTrackingCallbacks
 - el::base::Storage, 186
- m_performanceTrackingMap
 - el::base::TypedConfigurations, 208
- m_pFlags
 - el::base::VRegistry, 215
- m_preRollOutCallback

- el::base::Storage, 186
- m_proceed
 - el::base::LogDispatcher, 105
 - el::base::Writer, 219
- m_registeredHitCounters
 - el::base::Storage, 186
- m_registeredLoggers
 - el::base::Storage, 187
- m_resolver
 - el::CustomFormatSpecifier, 76
- m_stream
 - el::Logger, 122
- m_subsecondPrecisionMap
 - el::base::TypedConfigurations, 209
- m_termSupportsColor
 - el::LogBuilder, 99
- m_threadNames
 - el::base::Storage, 187
- m_threadNamesLock
 - el::base::Storage, 187
- m_toFileMap
 - el::base::TypedConfigurations, 209
- m_toStandardOutputMap
 - el::base::TypedConfigurations, 209
- m_typedConfigurations
 - el::Logger, 122
- m_unflushedCount
 - el::Logger, 122
- m_userFormat
 - el::base::LogFormat, 111
- m_value
 - el::Configuration, 60
- m_verboseLevel
 - el::base::Writer, 220
 - el::LogMessage, 135
- m_vRegistry
 - el::base::Storage, 187
- m_width
 - el::base::SubsecondPrecision, 195
- main
 - main.cpp, 365
- main.cpp
 - main, 365
- MAKE_CONTAINERELPP_FRIENDLY
 - easylogging++.h, 310
- MAKE_LOGGABLE
 - easylogging++.h, 311
- MaxLogFileSize
 - el, 17
- maxLogFileSize
 - el::base::TypedConfigurations, 204
- message
 - el::LogMessage, 134
- MessageBuilder
 - el::base::MessageBuilder, 136
- Microsecond
 - el::base, 23
- Millisecond
 - el::base, 23
- MillisecondsWidth
 - el, 17
 - el::base, 22
- millisecondsWidth
 - el::base::TypedConfigurations, 204
- Minute
 - el::base, 23
- modules
 - el::base::VRegistry, 213
- MultiLoggerSupport
 - el, 18
- Mutex
 - el::base::threading, 36
- name
 - el::base::consts, 34
- newFileStream
 - el::base::utils::File, 84
- NewLineForContainer
 - el, 18
- NoCopy
 - el::base::NoCopy, 139
- NoMutex
 - el::base::threading::internal::NoMutex, 140
- None
 - el::base, 22
- NormalLog
 - el::base, 22
- NoScopedLock
 - el::base::threading::internal::NoScopedLock < Mutex >, 142
- Not
 - el::base::utils::bitwise, 42
- NullWriter
 - el::base::NullWriter, 143
- numb
 - el::base::consts, 34
- operator bool
 - el::base::NullWriter, 143
 - el::base::Writer, 217
- operator!=
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 49
- operator<<
 - el::base::MessageBuilder, 137
 - el::base::NullWriter, 143
 - el::base::utils, 40
 - el::base::utils::CommandLineArgs, 56
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 174
 - el::base::Writer, 217, 218
 - el::Loggable, 113
- operator()
 - el::base::HitCounter::Predicate, 153
 - el::Configuration::Predicate, 154
 - std::hash< el::Level >, 86
- operator=

- el::base::HitCounter, 93
 - el::base::LogFormat, 109
 - el::base::NoCopy, 139
 - el::base::StaticClass, 177
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 49
 - el::base::utils::Registry< T_Ptr, T_Key >, 168
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 173
 - el::Configuration, 59
 - el::Logger, 118
- operator==
 - el::base::LogFormat, 109
 - el::base::SubsecondPrecision, 195
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 49
 - el::CustomFormatSpecifier, 76
- Or
 - el::base::utils::bitwise, 42
- ostream_t
 - el::base::type, 38
- parentApplicationName
 - el::Logger, 118
- parseFormat
 - el::base::utils::DateTime, 79
- parseFromFile
 - el::Configurations, 66
 - el::Configurations::Parser, 147
- parseFromFormat
 - el::base::LogFormat, 109
- parseFromText
 - el::Configurations, 66
 - el::Configurations::Parser, 148
- parseLine
 - el::Configurations::Parser, 148
- pathExists
 - el::base::utils::File, 85
- PCHECK
 - easylogging++.h, 311
- PERFORMANCE_CHECKPOINT
 - easylogging++.h, 311
- PERFORMANCE_CHECKPOINT_WITH_ID
 - easylogging++.h, 312
- PerformanceTrackerPtr
 - el::base::type, 38
- PerformanceTracking
 - el, 17
- performanceTracking
 - el::base::TypedConfigurations, 205
- PerformanceTrackingCallbackPtr
 - el::base::type, 38
- PErrorWriter
 - el::base::PErrorWriter, 152
- PLOG
 - easylogging++.h, 312
- PLOG_IF
 - easylogging++.h, 312
- populateAllLoggerIds
 - el::Loggers, 129
- Predicate
 - el::base::HitCounter::Predicate, 153
 - el::Configuration::Predicate, 154
- PreRollOutCallback
 - el, 17
- preRollOutCallback
 - el::base::Storage, 182
- processDispatch
 - el::base::Writer, 218
- README, 1
- README.md, 365
- reconfigure
 - el::Logger, 118
- reconfigureAllLoggers
 - el::Loggers, 129
- reconfigureLogger
 - el::Loggers, 130
- RegisteredLoggers
 - el::base::RegisteredLoggers, 161
- registeredLoggers
 - el::base::Storage, 182
- registerNew
 - el::base::utils::Registry< T_Ptr, T_Key >, 168
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 173
- Registry
 - el::base::utils::Registry< T_Ptr, T_Key >, 167
- RegistryWithPred
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 172
- reinitDeepCopy
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 50
- releaseDate
 - el::VersionInfo, 211
- releaseLock
 - el::base::threading::ThreadSafe, 198
- remove
 - el::base::RegisteredLoggers, 162
- removeFlag
 - el::base::Storage, 182
 - el::base::utils, 41
 - el::Loggers, 130
- replaceAll
 - el::base::utils::Str, 190, 191
- replaceFirstWithEscape
 - el::base::utils::Str, 191
- reserveCustomFormatSpecifiers
 - el::Helpers, 89
- resetLocation
 - el::base::HitCounter, 94
- resolveFilename
 - el::base::TypedConfigurations, 205
- resolveLoggerFormatSpec
 - el::Logger, 119
- resolver
 - el::CustomFormatSpecifier, 76

- rtrim
 - el::base::utils::Str, [191](#)
- safeDelete
 - el::base::utils, [41](#)
- ScopedAddFlag
 - el::Loggers::ScopedAddFlag, [174](#)
- ScopedLock
 - el::base::threading, [36](#)
- ScopedRemoveFlag
 - el::Loggers::ScopedRemoveFlag, [175](#)
- Second
 - el::base, [23](#)
- set
 - el::Configurations, [67](#)
- setApplicationArguments
 - el::base::Storage, [182](#)
- setArgs
 - el::base::utils::CommandLineArgs, [55](#)
 - el::Helpers, [89](#)
- setDefaultConfigurations
 - el::base::RegisteredLoggers, [163](#)
 - el::Loggers, [131](#)
- setDefaultLogBuilder
 - el::base::RegisteredLoggers, [163](#)
 - el::Loggers, [131](#)
- setDispatchAction
 - el::LogDispatchData, [103](#)
- setEnabled
 - el::Callback< T >, [52](#)
- setFlags
 - el::base::Storage, [182](#)
- setFromArgs
 - el::base::VRegistry, [213](#)
- setFromBase
 - el::Configurations, [67](#)
- setGlobally
 - el::Configurations, [68](#)
- setLevel
 - el::base::VRegistry, [214](#)
- setLogBuilder
 - el::Logger, [119](#)
- setLoggingLevel
 - el::base::Storage, [183](#)
 - el::Loggers, [131](#)
- setLogMessage
 - el::LogDispatchData, [103](#)
- setModules
 - el::base::VRegistry, [214](#)
- setParentApplicationName
 - el::Logger, [119](#)
- setPreRollOutCallback
 - el::base::Storage, [183](#)
- setRemainingToDefault
 - el::Configurations, [68](#)
- setStorage
 - el::Helpers, [89](#)
- setThreadName
 - el::base::Storage, [183](#)
- el::Helpers, [90](#)
- setDefault
 - el::Configurations, [69](#)
- setValue
 - el::base::TypedConfigurations, [205](#)
 - el::Configuration, [59](#)
- setVerboseLevel
 - el::Loggers, [131](#)
- setVModules
 - el::Loggers, [131](#)
- SHARE_EASYLOGGINGPP
 - easylogging++.h, [312](#)
- size
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, [50](#)
 - el::base::utils::CommandLineArgs, [55](#)
- src/main.cpp, [365](#), [366](#)
- START_EASYLOGGINGPP
 - easylogging++.h, [312](#)
- startsWith
 - el::base::utils::Str, [191](#)
- StaticClass
 - el::base::StaticClass, [177](#)
- std, [42](#)
- std::hash< el::Level >, [85](#)
- operator(), [86](#)
- Storage
 - el::base::Storage, [179](#)
- storage
 - el::Helpers, [90](#)
- StoragePointer
 - el::base::type, [38](#)
- STRCAT
 - easylogging++.h, [312](#)
- STRCPY
 - easylogging++.h, [313](#)
- stream
 - el::Logger, [119](#)
- STRERROR
 - easylogging++.h, [313](#)
- StrictLogFileSizeCheck
 - el, [18](#)
- string_t
 - el::base::type, [38](#)
- stringstream_t
 - el::base::type, [38](#)
- stringToLevelMap
 - el, [19](#)
- STRTok
 - easylogging++.h, [313](#)
- SubsecondPrecision
 - el, [17](#)
 - el::base::SubsecondPrecision, [194](#)
- subsecondPrecision
 - el::base::TypedConfigurations, [205](#)
- SYSLOG
 - easylogging++.h, [313](#)
- SysLog

- el::base, 22
- SYSLOG_AFTER_N
 - easylogging++.h, 313
- SYSLOG_EVERY_N
 - easylogging++.h, 313
- SYSLOG_IF
 - easylogging++.h, 314
- SYSLOG_N_TIMES
 - easylogging++.h, 314
- SysLogInitializer
 - el::SysLogInitializer, 196
- termSupportsColor
 - el::base::utils::OS, 146
- ThreadId
 - el::base, 23
- ThreadSafe
 - el::base::threading::ThreadSafe, 198
- TIMED_BLOCK
 - easylogging++.h, 314
- TIMED_FUNC
 - easylogging++.h, 314
- TIMED_FUNC_IF
 - easylogging++.h, 314
- TIMED_SCOPE
 - easylogging++.h, 315
- TIMED_SCOPE_IF
 - easylogging++.h, 315
- TimestampUnit
 - el::base, 23
- timevalToString
 - el::base::utils::DateTime, 79
- Todo List, 3
- ToFile
 - el, 17
- toFile
 - el::base::TypedConfigurations, 205
- ToStandardOutput
 - el, 17
- toStandardOutput
 - el::base::TypedConfigurations, 206
- toUpper
 - el::base::utils::Str, 192
- Trace
 - el, 18
- triggerDispatch
 - el::base::Writer, 218
- trim
 - el::base::utils::Str, 192
- try_lock
 - el::base::threading::internal::NoMutex, 140
- TypedConfigurations
 - el::base::TypedConfigurations, 201, 202
- typedConfigurations
 - el::Logger, 119
- uninstallCallback
 - el::base::utils::Utils, 210
- uninstallCustomFormatSpecifier
 - el::base::Storage, 183
- el::Helpers, 90
- uninstallLogDispatchCallback
 - el::base::Storage, 183
 - el::Helpers, 90
- uninstallLoggerRegistrationCallback
 - el::base::RegisteredLoggers, 163
 - el::Loggers, 132
- uninstallPreRollOutCallback
 - el::Helpers, 91
- unit
 - el::base::consts, 35
- Unknown
 - el, 17, 18
- unlock
 - el::base::threading::internal::NoMutex, 140
- unregister
 - el::base::RegisteredLoggers, 163
 - el::base::utils::Registry< T_Ptr, T_Key >, 168
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 173
- unregisterAll
 - el::base::utils::AbstractRegistry< T_Ptr, Container >, 50
 - el::base::utils::Registry< T_Ptr, T_Key >, 169
 - el::base::utils::RegistryWithPred< T_Ptr, Pred >, 173
- unregisterLogger
 - el::Loggers, 132
- unsafeFlushAll
 - el::base::RegisteredLoggers, 163
- unsafeGetConfigByRef
 - el::base::TypedConfigurations, 206
- unsafeGetConfigByVal
 - el::base::TypedConfigurations, 206
- unsafeSet
 - el::Configurations, 69
- unsafeSetGlobally
 - el::Configurations, 69
- unsafeSetIfNotExist
 - el::Configurations, 70
- unsafeValidateFileRolling
 - el::base::TypedConfigurations, 206
- unsetPreRollOutCallback
 - el::base::Storage, 183
- updateDateFormat
 - el::base::LogFormat, 110
- updateFormatSpec
 - el::base::LogFormat, 110
- User
 - el::base, 22
- userFormat
 - el::base::LogFormat, 110
- validateAfterN
 - el::base::RegisteredHitCounters, 157
- validateAfterNCounter
 - el::base::Storage, 184
- validateEveryN

- el::base::RegisteredHitCounters, 157
- validateEveryNCounter
 - el::base::Storage, 184
- validateFileRolling
 - el::base::TypedConfigurations, 206
 - el::Helpers, 91
- validateHitCounts
 - el::base::HitCounter, 94
- validateNTimes
 - el::base::RegisteredHitCounters, 158
- validateNTimesCounter
 - el::base::Storage, 184
- value
 - el::base::consts, 35
 - el::Configuration, 59
- Verbose
 - el, 18
- VerboseLevel
 - el::base, 23
 - el::base::type, 39
- verboseLevel
 - el::Loggers, 132
 - el::LogMessage, 134
- version
 - el::VersionInfo, 211
- VLOG
 - easylogging++.h, 315
- VLOG_AFTER_N
 - easylogging++.h, 315
- VLOG_EVERY_N
 - easylogging++.h, 316
- VLOG_IF
 - easylogging++.h, 316
- VLOG_IS_ON
 - easylogging++.h, 316
- VLOG_N_TIMES
 - easylogging++.h, 316
- vModulesEnabled
 - el::base::VRegistry, 214
- VRegistry
 - el::base::VRegistry, 213
- vRegistry
 - el::base::Storage, 184
- Warning
 - el, 18
- wcharPtrToCharPtr
 - el::base::utils::Str, 192
- wildCardMatch
 - el::base::utils::Str, 192
- WIP, 42
 - exampleEasyLogging, 43
- writelIterator
 - el::base::MessageBuilder, 137
- Writer
 - el::base::Writer, 216