Documentation of MaiLogger v. 3.0

Bernhard Gaida

March 14, 2021

Contents

1 Intro

MaiLogger is a tool for logging events representing the state of a program. MaiLogger supports following features

- Logging 5 groups of events (debug, info, warning, error, critical) + any user-specified groups (see: Logging)

- writing logged events to a file

- Setting the folder of that file (see: Configure Mailogger)

- Loading log files (see: Load a log file)

- Setting the maximum number of events to log (deletes old ones in case of two many events; uses FIFO manner to remove log entries) (see: Configure Mailogger)

- Log rotation (see: Log Rotation)

- Setting the maximum number of files saved by the log rotation (see: Configure Mailogger)

- Setting whether the time representing the occurrence of an event is also logged (see: Configure Mailogger)

- Stopping the application if a critical error occurs (see: Log critical)

- Logging of tasks with the appendix of "success", "failed" or "abort" (see: Tasks)

All classes of MaiLogger are located in the package logger. In the following we will ignore the reference to this package and will only use the names of those classes. (e.g: logger.MaiLogger -> MaiLogger)

1.1 Log file

Every logged event will be written to a log file. This file is located in the directory which has been determined (see: Configure MaiLogger). The name of that file is the name of the class which implements the interface MaiLog (written in lowercase). This file is also used to load log files by calling the method `MaiLogger.load()`.

2   Setup

1. Recommended: Implement the Interface `MaiLog`

2. Call `MaiLogger.setUp(...)`

2.1   Implement the Interface MaiLog

Implement the Interface `MaiLog` with the three methods `stop()`, `sendLog(String)` and `sendErrMsg(String)`. The class implementing this interface should be able to stop the application. If there is no use in the methods implemented by the interface MaiLog you can skip implementing this interface.

stop():
This method will be called as soon as a critical error occurs. This means, that as soon as the method `MaiLogger.logCritical(String)`is called, the event responsible for the critical state of the application will be logged and the method `MaiLog.stop()` will be called afterwards. The purpose of this method is to stop the application when it reaches a critical state.

sendLog(String):
This method will be called every time an event is logged, means that one of the following methods is called:

- `MaiLogger.logDebug (String)`

- `MaiLogger.logInfo (String)`

- `MaiLogger.logWarning (String)`

- `MaiLogger.logError (String)`

- `MaiLogger.logCritical(String)`

- `MaiLogger.logCustom (String, String)`

- `MaiLogger.logNewTask (String)`

- `MaiLogger.succeededTask (int)`

- `MaiLogger.failedTask (int)`

- `MaiLogger.on_exit ()`

The argument of this method is the description of the event. The developer is free to choose what to do with this information. The events will be saved in the log files independent of the implementation of this method and the description of the event handed over in `sendLog(String)` and saved in the log files are identical

sendErrMsg(String):
This method is called if MaiLogger runs into an error. There are several reasons for this, but often it's an I/O Error. The argument of this method is the description of the event reasoning for this error. The following table gives an overview about possible reasons and the handed-over description

Called Method of class MaiLogger	Reason	Description
`succeededTask(int)` `failedTask (int)`	Task with the id (1st argument) does not exist	Task with ID " + id + " was not found
`save()`	The directory in which the log files should be stored cannot be created	ERROR: Cannot write logs to file: Cannot create directory: ...
`save()`	I/O Error while writing events to file or while creating/removing logfile.backup	ERROR: Cannot write logs to file ...
`load()`	The file to load does not exist	ERROR: Cannot load logfile: File does not exist
`load()`	I/O Error while reading the file	ERROR: Cannot load logfile
`load()`	Invalid description of a logged event	ERROR: Cannot load logfile
`rotate()`	The directory with the files to rotate is empty	ERROR: Cannot rotate: No files in directory
`rotate()`	It isn't possible to rename the files which is necessary for the rotation	ERROR: Cannot rotate: Unable to rename files
`rotate()`	The directory with the files is empty after rotating	ERROR: Cannot rotate: No files in directory after rotating
`rotate()`	I/O Error occurred while deleting old files	"ERROR: Cannot delete file

These errors does not cause MaiLogger to exit, but it will may be limited by its functionality.

2.2 Configure MaiLogger - Call MaiLogger.setUp(...)

Before using one of the method to log an event Mailogger must be configured. There are three methods supporting this:

```
MaiLogger.setUp(MaiLog mainclass, int maxlogs, int rotations, boolean time, boolean ImmediateOutput,
    String directory, String logFileName)
```

```
MaiLogger.setUp(int maxlogs, int rotations, boolean time, boolean ImmediateOutput, String directory,
    logFileName)
```

```
MaiLogger.setUp(MaiLog mainclass)
```

If no of these methods is called MaiLogger will use the default settings.

2.2.1 Parameters

MaiLogger.setUp(...) expects up to the following arguments. Each argument which is not expected is set to default.

MaiLog mainclass: The class implementing the interface MaiLog (see: Implement MaiLog). If mainclass is null, MaiLogger.setUp(...) will through a NullPointerException

int maxLogs: Sets the maximum number of logged events in the log file. If the maximum number of logged ones is reached the oldest event will be removed when logging a new one. Set maxLogs to -1 to not limit the maximum number of logged events. It is recommend to set it to -1 to make MaiLogger work more efficient. The default value is -1.

int rotations: Sets the maximum number of files created by log rotation. Set this value to -1 to not limit the maximum number of files. The default value is -1. (see: Log rotation)

boolean time: Decides whether the time when the event has been logged is also logged. Set it to true to log the time. If "time" is true, every log entry will have the following format:

dd.MM.yy HH:mm:ss <group>:<event>

The default value is true.

boolean ImmediateOutput: Decides whether tasks will be logged as soon as they start and the status (success/fail/abort) will be logged afterwards or whether tasks will wait until they have been finished or aborted. Set it to true to immediately log tasks. The default value is true.

String directory:
Sets the path to the directory which the log files will be written to. Be sure the directory exists otherwise create it. To use the default value set "directory" to "". The default value is "/../logs"

String logFileName:
Sets the name of the file where all the logs are written to. The default value is the name of the class mainClass or "Log" if there if the method setUp is not called.

3   On exit

When exiting the application it is highly recommend to call `MaiLogger.on_exit()`. This makes sure that every task is logged before exiting. This method logs every running task with the adding "ABORTED BY USER". If tasks are not used by the program, there is no need to call `MaiLogger.on_exit()`.

4   Logging

MaiLogger supports six methods to log an event:

- `MaiLogger.logDebug (String)`

- `MaiLogger.logInfo (String)`

- `MaiLogger.logWarning (String)`

- `MaiLogger.logError (String)`

- `MaiLogger.logCritical(String)`

- `MaiLogger.logCustom (String, String)`

Every method assigns the event to another group.

4.1   Groups

There are 5 fix groups an event can be assigned to: debug, info, warning, error, critical. Every group represents another priority of the logged events:

Debug:
This group represents information useful for debugging.

info:
This group represents a generally useful information which can be used to locate the programs state (e.g: start/stop of a service). Events of this group usually represent an expected behavior.

warning:
This group represents events that can potentially cause a deviation in the expected behavior of the program.
These events can lead to an error or even a critical state of the program.

error:
This group represents events that are fatal to an operation, but not to the application (e.g. Inability to open a file) These events can lead to a critical state of the program.

critical:
This group represents events that are forcing a shutdown of the program to prevent data loss or because of the inability to run the program as expected.

4.2   Log Debug

The method `MaiLogger.logInfo (String msg)` logs an event to the group debug. The first argument of the method is the description of the event. The log entry has the following format:

DEBUG: <msg>

or

dd.MM.yy HH:mm:ss DEBUG: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument.

4.3   Log Info

The method `MaiLogger.logInfo (String msg)` logs an event to the group info. The first argument of the method is the description of the event. The log entry has the following format:

INFO: <msg>

or

dd.MM.yy HH:mm:ss INFO: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument.

4.4   Log Warning

The method `MaiLogger.logWarning (String msg)` logs an event to the group warning. The first argument of the method is the description of the event. The log entry has the following format:

WARNING: <msg>

or

dd.MM.yy HH:mm:ss WARNING: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument.

4.5 Log Error

The method `MaiLogger.logError (String msg)` logs an event to the group error. The first argument of the method is the description of the event. The log entry has the following format:

ERROR: <msg>

or

dd.MM.yy HH:mm:ss ERROR: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument.

4.6 Log Critical

The method `MaiLogger.logCritical (String msg)` logs an event to the group critical. The first argument of the method is the description of the event. The log entry has the following format:

CRITICAL: <msg>

or

dd.MM.yy HH:mm:ss CRITICAL: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument. Afterwards all running tasks will be logged with the adding "ABORTED BY CRITICAL STATE" and the termination of the application will be triggered by calling `MaiLog.stop()`.

4.7 Log Custom

The method `MaiLogger.logInfo (String msg, String group)` can be used to log an event to a user-defined group. It logs an event to that user-defined group. The first argument of the method is the description of the event. The second argument of the method is the name of the group. The log entry has the following format:

<group>: <msg>

or

dd.MM.yy HH:mm:ss <group>: <msg>

This entry will be written to the end of the log file and in addition the method `MaiLog.sendLog(String)` is called with this entry handed-over as argument.

5 Tasks

In MaiLogger a task is an event that represents a task in a program which can succeed, fail or be aborted. The purpose of tasks in MaiLogger is to log only one entry with the adding success, failed or aborted and the time the task needed to finish, instead of two entries for starting and stopping the task. For this MaiLogger supports special methods to log tasks:

- `MaiLogger.logNewTask (String)`

- `MaiLogger.succeededTask (int)`

- `MaiLogger.failedTask (int)`

5.1 Log New Task

The method `logTask(String msg)` creates a new event representing a task where the first argument represents the description of that task. The return value of this method is the id representing this event. This event is set as running task and is logged using the group task. The log entry has the following format:

TASK: <msg> ...

or

dd.MM.yy HH:mm:ss TASK: <msg> ...

To tell a task as finished, call `succeededTask(int id)` or `failedTask(int id)`. Whenever a critical event was logged, all tasks will be logged with the adding "ABORTED BY CRITICAL STATE". When the method `MaiLogger.on_exit()` is called, every task will be logged with the adding "ABORTED BY USER". As soon as a task has finished or has been aborted, it is no longer possible to change its status.

5.2 Succeeded Task

The method `succeededTask(int id)` sets the task with the id, handed-over as argument, as successfully finished. Furthermore it adds the suffix "SUCCESS" to the existing log entry of this task with the time needed to complete the task. If another log was in between it logs the task as a new event of the group info instead and adds the time needed to complete the task and the notice that the task finished successfully. The final log entry has the following format:
If no log was in between:

TASK: <msg> ... SUCCESS (<runtime>)

or

dd.MM.yy HH:mm:ss TASK: <msg> ... SUCCESS (<runtime>)

If logs were in between:

TASK: <msg> ...

... some other logs ...

INFO: <msg> ... SUCCESS (<runtime>)

or

dd.MM.yy HH:mm:ss TASK: <msg> ...

... some other logs ...

dd.MM.yy HH:mm:ss INFO: <msg> ... SUCCESS (<runtime>)

(Here, the time is specified, in which the task was created)
If the task with the id doesn't exist, a new log entry will be created with the information that the task wasn't found and logged as event of the group "error". Furthermore `MaiLog.sendErrMsg(String)` will be called with this information.

8

5.3 Failed Task

The method `failedTask(int id)` sets the task with the id, handed-over as argument, as failed. Furthermore it adds the suffix "FAILED" to the existing log entry of this task with the time needed to complete the task if there were no other logs in between. In addition, it logs the task as event of the group error and adds the time needed to complete the task and the notice that the task failed. The final log entry has the following format:
If no log was in between:

TASK: <msg> ... FAILED (<runtime>)

ERROR: <msg> ... FAILED (<runtime>)

or

dd.MM.yy HH:mm:ss TASK: <msg> ... FAILED (<runtime>)

dd.MM.yy HH:mm:ss ERROR: <msg> ... FAILED (<runtime>)

If logs were in between:

TASK: <msg> ...

... some other logs ...

ERROR: <msg> ... SUCCESS (<runtime>)

or

dd.MM.yy HH:mm:ss TASK: <msg> ...

... some other logs ...

dd.MM.yy HH:mm:ss ERROR: <msg> ... SUCCESS (<runtime>)

(Here, the time is specified, in which the task was created)
If the task with the id doesn't exist, a new log entry will be created with the information that the task wasn't found and logged as event of the group "error". Furthermore `MaiLog.sendErrMsg(String)` will be called with this information.

6 Log Rotation

To rotate the log files, MaiLogger provides the method `MaiLogger.rotate()` Rotation means that the current log file (here,all log files are called <mainclass>)

<mainclass>.log is renamed to <mainclass>.log.1,

<mainclass>.log.1 is renamed to <mainclass>.log.2 if <mainclass>.log.1 exists,

<mainclass>.log.2 is renamed to <mainclass>.log.3 if <mainclass>.log.2 exists

...

Afterwards the current logged events will be removed and a new empty log file <mainclass>.log will be created. If a maximum number of log files has been set (see: Configure MaiLogger) and exceeded by a log rotation, the oldest file will deleted.

9

7   Further features

7.1   Load a log file

MaiLogger provides a method `MaiLogger.load()` to load a log file. This method loads all log entries of the log file stored in the directory specified by the Configuration of MaiLogger (see: Configure MaiLogger). This will override all current existing log entries. If Mailogger hasn't been set up yet, MaiLogger.load() will throw an IllegalStateException.

7.2   Get the log entries

MaiLogger provides two methods which return the current log entries:

- `MaiLogger.getLog(boolean info, boolean warning, boolean error, boolean critical)`

- `MaiLogger.getLogAll()`

MaiLogger.getLog(int ...):
This method expects several arguments, where each one represents the group of which the log entries should be returned.
E.g:

  To get all log entries of the group "info" then add the argument Group.INFO:
      `MaiLogger.getLog(Group.INFO)`

  To get all log entries of the group "warning" then add the argument Group.WARNING:
      `MaiLogger.getLog(Group.WARNING)`

  To get all log entries of the group "info" and "error" then add the arguments Group.INFO and Group.WARNING:
      `MaiLogger.getLog(Group.INFO, Group.ERROR)`

$$\vdots$$

To get log entries of a user-defined group use the group Group.CUSTOM.
All log entries will be returned in chronologically order.

MaiLogger.getLogAll():
This method returns all log entries in chronologically order. It is equals to `MaiLogger.getLog(true, true, true, true)`

7.3   Clear log

MaiLogger provides a method to remove all current log entries and tasks. This does not affect the log file.