SimpleLogging Package

Version 1.8
December 2021

Gray Watson

Table of Contents

SimpleLogging 1

1 Start Using Quickly 2

2 Using SimpleLogging 3

2.1 Downloading Jar 3
2.2 Using SimpleLogging in Your Library 3
2.3 How SimpleLogging Discovers the Logging Backlend 3
2.4 Setting Log Level or Disabling Logging 4
2.5 More Usage Examples 4
2.6 Using With Maven 5

3 Open Source License 6

Index of Concepts 7

SimpleLogging

Version 1.8 – December 2021

The goal is a small-ish library that backends to a number of standard logging packages. This allows you to write your code and include log messages without having a fixed dependency on any one logging package. I include this code into my libraries and so they can stay agnostic. I understand that this is the goal of commons-logging as well but I really want the slf4j-style `{}` arguments which aren't supported. This logging code allows you to write messages with the `{}` support and then backend to a number of other logging libraries or you can easily implement your own. It also handles arrays appropriately and supports up to 4 arguments before forcing the caller to pass in an object array or calling a different `logArgs(...)` method for variable arguments.

To get started quickly using SimpleLogging, see Chapter 1 [Quick Start], page 2. There is also a HTML version of this documentation.

Gray Watson `http://256stuff.com/gray/`

1 Start Using Quickly

To use SimpleLogging in your code is similar to other logging libraries that you are used to.

```
// usually a logger will be per-class
// getLogger() also can take a String label
private static final Logger logger =
   LoggerFactory.getLogger(MyClass.class);
...

// log trace message with arguments
// toString() on the args only called if trace messages enabled
logger.trace("some trace information: {} and {}", arg1, arg2);
...

// NOTE: exception argument comes _before_ the
// message format to not confuse the arguments
logger.error(exception, "http client threw getting URL: {}", url);
```

For somewhat more extensive instructions, see Chapter 2 [Using], page 3.

2 Using SimpleLogging

2.1 Downloading Jar

To get started with SimpleLogging, you will need to download the jar file. The SimpleLogging release page is the default repository but the jars are also available from the central maven repository.

The code works with Java 6 or later.

2.2 Using SimpleLogging in Your Library

To use the SimpleLogging code in your library, just copy the classes into a logging package in your code and rename the package as necessary. You may want to tune the order of discovery in the `LogBackendType` enumerated class or remove support for certain loggers that your library doesn't need support for.

2.3 How SimpleLogging Discovers the Logging Backlend

Built into the `LoggerFactory` class is a `LogBackendType` enumerated type which SimpleLogging uses to try to locate what logging packages are on the classpath and therefore should be used.

The following logging implementations will be discovered on the classpath in this order.

1. SLF4J – (often paired with logback)
2. ANDROID – Android native Log
3. LOGBACK – Logback direct without SLF4J
4. COMMONS_LOGGING – Apache Commons Logging
5. LOG4J2 – version 2+
6. LOG4J – older
7. LAMBDA – AWS Lambda systems logging
8. LOCAL – log implementation that can write to a simple file
9. CONSOLE – log writing to `System.out` or `System.err`
10. JAVA_UTIL – Java util logging which is usually available in the JRE but never chosen directly
11. NULL – null logger to log no messages

If you (or the users of your library) want to force the logging type, they can use the following static methods.

```
// used to set a specific LoggerFactory.LogBackendType
LoggerFactory.setLogBackendType(LogBackendType.LOGBACK);
```

Or to specify a specifically factory class including custom ones:

```
// used to set a specific LoggerFactory or custom
LoggerFactory.setLogBackendFactory(new LogbackLogBackendFactory());
```

You can also set the `com.j256.simplelogger.backend` Java property to one of the `LogBackendType` values in the above list. For example: `java -Dcom.j256.simplelogger.backend=LOGBACK`

2.4 Setting Log Level or Disabling Logging

By default the choice of whether or not to log, for example, a trace log message is up to the specific log backend. Your LOG4J settings determine whether or not trace logging is enabled for a particular package. However there are some ways that you can impact which messages will get logged with calls to the SimpleLogging methods.

```
// have all INFO or above messages to be logged
// regardless of the backend configurations
Logger.setGlobalLogLevel(Level.INFO);
```

You can also disable all log messages by using the NULL level:

```
// force all messages to be disabled
Logger.setGlobalLogLevel(Level.NULL);
```

Another way to not have any log messages show would be use use the `NULL` log backend.

```
// set the backend type to be the null logger
LoggerFactory.setLogBackendType(LogBackendType.NULL);
```

2.5 More Usage Examples

Usually the logger is defined per-class as a `static` field.

```
private static final Logger logger =
   LoggerFactory.getLogger(MyClass.class);
```

The logger handles trace, debug, info, warn, error, and fatal messages. Not all of these message types are supported by all of the logger backends so SimpleLogging tries to map where appropriate.

```
logger.trace("read '{}' bytes", readCount);
logger.debug("host '{}', port-number {}", host, port);
logger.info("connecting to: {}:{}", host, port);
logger.warn("retry connect to host '{}', port {}", host, port);
logger.error("connect failed to host '{}', port {}", host, port);
logger.fatal(exception, "host '{}' threw exception", host);
```

If you want to use more than 3 arguments, you will either need to pass in an `Object[]` or call the methods with the `Args` suffix. This is a specific design decision because in either case, an `Object[]` is created even if the message is never logged.

```
logger.debug("scheme '{}', host '{}', port {}, path: {}",
     new Object[] { schema, host, port, path });
```

Or with variable arguments with methods that have an `Args` suffix such as `debugArgs(...)`.

```
logger.debugArgs("scheme '{}', host '{}', port {}, path: {}",
     schema, host, port, path);
```

2.6 Using With Maven

To use SimpleLogging with maven, include the following dependency in your '`pom.xml`' file:

```
<dependency>
   <groupId>com.j256.simplelogging</groupId>
   <artifactId>simplelogging</artifactId>
   <version>1.8</version>
</dependency>
```

3 Open Source License

Index of Concepts

A

android logging, use with . . . 3
author . . . 1
aws lambda logging, use with . . . 3

C

commons logging, use with . . . 3
copying code . . . 3

D

disable all messages . . . 4
discovering backend . . . 3
downloading the jars . . . 3

G

getting started . . . 2

H

how to download the jars . . . 3
how to get started . . . 2
how to use . . . 3

I

introduction . . . 1
isc license . . . 6

J

java util logging, use with . . . 3

L

lambda logging, use with . . . 3
license . . . 6
log backend discovery . . . 3
log level, setting . . . 4
log type of backends . . . 3
log4j, use with . . . 3
log4j2, use with . . . 3
logback direct, use with . . . 3
Logger . . . 2

M

Maven, use with . . . 5

O

open source license . . . 6

P

pom.xml dependency . . . 5

Q

quick start . . . 2

S

setting log level . . . 4
simple logging . . . 1
slf4j, use with . . . 3

U

using in your library . . . 3
using SimpleLogging . . . 3

V

varargs, use with . . . 4
variable arguments, use with . . . 4

W

where to get new jars . . . 3