Frequently Asked Questions

Logback project

Why is logback distributed under LGPL and not the Apache Software License (ASL)?

The logback project is dual licensed under the LGPL and the EPL for two main reasons. For one, the different license emphasizes that the fact that logback is a related but *different* project than log4j.

Both the LGPL and EPL are reasonable and widely-accepted licenses. In contrast to the ASL, both the LGPL and he EPL require that derivate work be licensed under the same license. While there might be debate about the exact definition of derivative work, we find such reciprocity both justified and morally appealing -- that is the second reason for our choice of the LGPL & EPL dual-license. The subtly more liberal approach embodied in the ASL is not necessarily wrong. It is the expression of a different balance.

What are logback's dependencies, i.e. JDK version and third-party libraries?

This question is answered on a separate page dedicated to the question of dependencies.

Logback-classic

How does the automatic configuration work?

This question is answered in the relevant section of the logback manual.

Where should the configuration files such as *logback.groovy*, *logback-test.xml* or *logback.xml* be located on the classpath?

Configuration files such as *logback.groovy*, *logback-test.xml* or *logback.xml* can be located **directly** under any folder declared in the class path. For example, if the class path reads "c:/java/jdk15/lib/rt.jar;c:/mylibs/" then the *logback.xml* file should be located directly under "c:/mylibs/", that is as "c:/mylibs/logback.xml". Placing it under a sub-folder of c:/mylibs/, say, c:/mylibs/other/, will not work.

For web-applications, configuration files can be placed **directly** under *WEB-INF/classes/*.

Are logback loggers serializable?

Yes. A logback logger *is* an SLF4J logger and SLF4J loggers are serializable. This means that an object referencing a logger will be able to log after its deserialization.

The deserialized logger instance will be generated by `org.slf4j.LoggerFactory`. Thus, it is possible for a logback logger to be deserialized as a log4j or j.u.l. logger, depending on the deserialization environment.

Is it possible for multiple JEE applications to share the same configuration file but without stepping on each other's toes?

Yes, it is. Using variable substitution, it is possible to have a single configuration file to output logs to different destinations for each JEE application. Here is a sample configuration file designed for this purpose.

```
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <!-- "application-name" is a variable -->
    <File>c:/logs/${application-name}.log</File>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>%d %p %t %c - %m%n</Pattern>
    </layout>
  </appender>
  <root level="debug">
    <appender-ref ref="FILE"/>
  </root>
</configuration>
```

Assuming each JEE application loads a different copy of logback classes into memory, if we can somehow inject a different value for *`application-name`* each time an application starts, logs will be output to different files. We just need to initialize logback with the above configuration file while injecting a different value for *`application-name`* variable. Here is sample code that programmatically configures logback. It should be invoked during the initialization of your JEE applications.

```
  LoggerContext context = (LoggerContext) LoggerFactory.getILoggerFactory();
  JoranConfigurator jc = new JoranConfigurator();
  jc.setContext(context);
  context.reset(); // override default configuration
  // inject the name of the current application as "application-name"
  // property of the LoggerContext
  context.putProperty("application-name", NAME_OF_CURRENT_APPLICATION);
  jc.doConfigure("/path/to/the/above/configuration/file.xml");
```

How can I disable logging from the command line?

Logback does not allow logging to be disabled from the command line. However, if the configuration file allows it, you can set the level of loggers on the command line via a Java system property. Here is such a configuration file.

```
<configuration>
  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>%d [%thread] %level %logger - %m%n</Pattern>
    </layout>
  </appender>
  <root level="${root-level:-INFO}">
```

```
    <appender-ref ref="CONSOLE"/>
  </root>
</configuration>
```

Making use of variable substitution as well as default values for variables, if the *root-level* system property is set to OFF, then all logging will be turned off. However, if it is not set, it will assume the default value of INFO. Note that you can set the *root-level* system property to any level value of your choice. The value OFF is just an example.

How can Jetty be instructed to use logback-classic as its logging implementation?

The Jetty application server uses SLF4J for its internal logging.

Logback jar files must be present on Jetty's class path. These files are *logback-core-1.1.3.jar* and *logback-classic-1.1.3.jar*. These files should be placed under the *$JETTY_HOME/lib* directory.

Since Jetty uses an older version of SLF4J internally, we recommend that the old version be replaced by *slf4j-api-1.7.7.jar*. This file can be downloaded from the SLF4J project.

For automatically configuring logback-classic, you can place the file *logback.xml* under the *$JETTY_HOME/resources* directory. You can find sample configuration files in the *logback-examples/src/main/java/chapters/appenders/conf/* folder shipping within the logback distribution.
