

## 2.3. Logging Framework

### Chapter 2. Configuration Framework

[Prev](#)[Next](#)

## 2.3. Logging Framework

- 2.3.1. [Disabling Logging](#)
- 2.3.2. [Logging using Apache Log4J](#)
- 2.3.3. [Logging using JDK 1.4 java.util.logging](#)
- 2.3.4. [Logging using Simple Log](#)
- 2.3.5. [Logging using a Custom Log](#)

Logging is very important for debugging and identifying performance hot spots in an application, as well as getting a sense of how Kodo operates. Using logging is essential for developing any persistent classes with custom object-relational mapping extensions, since observing the SQL that is generated will assist in quickly identify any misconfigurations in the JDO metadata file. Kodo provides a very flexible logging system that integrates with many existing runtime systems, such as application servers and servlet runners.

Kodo JDO uses the [Apache Jakarta Commons Logging](#) thin library for issuing log messages. The Commons Logging libraries act as a wrapper around a number of popular logging APIs, including the [Jakarta Log4J](#) project, and the native [java.util.logging](#) package in JDK 1.4. If neither of these libraries are available, then logging will fall back to using a very simple console logging. The remainder of this section presumes that `Log4J` will be used for logging. For details on customization of the Commons project, or on details on any of the underlying logging packages, please see the appropriate project page.



#### Warning

*Logging can have a very serious performance impact on Kodo. Disable verbose logging (such as logging of SQL statements) before running any performance tests. It is advisable to limit or disable logging completely for a production system.*

Logging is done over a number of logging channels, each of which has a logging level, which controls the verbosity of log messages that are sent to the channel. Following is an overview of the logging channels that Kodo will use, with a summary of the different levels to which log messages will be sent.

- [com.solarmetric.kodo.MetaData](#): Information about the parsing of JDO metadata will be sent to the `trace` level of this channel. Warnings about potential problems with metadata will be sent to the `warn` channel.
- [com.solarmetric.kodo.Enhance](#): Messages issued by the JDO enhancer will be sent to this logger, on a variety of channels.
- [com.solarmetric.kodo.Runtime](#): General Kodo runtime messages will be sent to this channel.
- [com.solarmetric.kodo.Configuration](#): Information about Kodo Configuration will be sent to this channel.
- [com.solarmetric.kodo.Performance](#): Information about possible performance optimizations will be sent to the `trace` level of this channel.
- [com.solarmetric.kodo.impl.jdbc.JDBC](#): JDBC connection information will be sent to this channel.
- [com.solarmetric.kodo.impl.jdbc.SQL](#): This is the most common logging channel to use. Detailed information about the execution of SQL statements and connections will be sent to the `trace` channel. It is useful to enable this channel if you are curious about the exact SQL that Kodo issues to the data store.

[Prev](#)

2.2. Kodo JDO Properties

[Home](#)

Chapter 3. Creating Persistent Classes

**Note**

Verbose SQL information is sent to this channel only when using Kodo's own pooling DataSource implementation. When using a custom DataSource, consult the documentation for that DataSource for details on how to enable logging messages.

- [com.solarmetric.kodo.impl.jdbc.Schema](#): Details about the operation of the SchemaTool will be sent to this logging channel.

### 2.3.1. Disabling Logging

Disabling logging can be useful for performance analysis without any I/O overhead or to reduce verbosity at the console. To do this, set the `org.apache.commons.logging.Log` to `org.apache.commons.logging.impl.NoOpLog`. To do this via command line:

```
java -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.NoOpLog mypkg.MyClass
```

However, disabling logging permanently will cause all error messages to be consumed. So, we recommend using one of the more sophisticated mechanisms described below.

**Note**

Versions of the Apache Commons Logging prior to 1.0.3 ignore the `org.apache.commons.logging.Log` system property. To resolve this, upgrade to a more recent version of the logging APIs.

### 2.3.2. Logging using Apache Log4J

When Apache Log4J jars are present, the Commons Logging package will use it by default. In a standalone application, logging levels are controlled by a resource named `log4j.properties`, which should be available as a top-level resource (either at the top level of a jar file, or in the root of one of the `CLASSPATH` environment variable). When deploying to a web or EJB application server, Log4J configuration is often performed in a `log4j.xml` file instead of in a properties file. For further details on configuring Log4J, please see the [Log4J Manual](#).

Following are example `log4j.properties` files for configuring logging levels for Kodo.

**Example 2.4. Example log4j.properties file for moderately verbose logging**

```
log4j.rootCategory=WARN, console
log4j.category.com.solarmetric.kodo.impl.jdbc.SQL=WARN, console
log4j.category.com.solarmetric.kodo.impl.jdbc.JDBC=WARN, console
log4j.category.com.solarmetric.kodo.impl.jdbc.Schema=INFO, console
log4j.category.com.solarmetric.kodo.Performance=INFO, console
log4j.category.com.solarmetric.kodo.Metadata=WARN, console
log4j.category.com.solarmetric.kodo.Enhance=WARN, console
log4j.category.com.solarmetric.kodo.Query=WARN, console
log4j.category.com.solarmetric.kodo.Runtime=INFO, console

log4j.appender.console=org.apache.log4j.ConsoleAppender
```

[Prev](#)

2.2. Kodo JDO Properties

[Example 2.5. Example log4j.properties file for disabled logging](#)[Home](#)

Chapter 3. Creating Persistent Classes

```
log4j.rootCategory=ERROR, console
log4j.category.com.solarmetric.kodo.impl.jdbc.SQL=ERROR, console
log4j.category.com.solarmetric.kodo.impl.jdbc.JDBC=ERROR, console
log4j.category.com.solarmetric.kodo.impl.jdbc.Schema=ERROR, console
log4j.category.com.solarmetric.kodo.Performance=ERROR, console
log4j.category.com.solarmetric.kodo.MetaData=ERROR, console
log4j.category.com.solarmetric.kodo.Enhance=ERROR, console
log4j.category.com.solarmetric.kodo.Query=ERROR, console
log4j.category.com.solarmetric.kodo.Runtime=ERROR, console

log4j.appender.console=org.apache.log4j.ConsoleAppender
```

**Example 2.6. Example log4j.properties file for debugging logging**

```
log4j.rootCategory=TRACE, console
log4j.category.com.solarmetric.kodo.impl.jdbc.SQL=TRACE, console
log4j.category.com.solarmetric.kodo.impl.jdbc.JDBC=TRACE, console
log4j.category.com.solarmetric.kodo.impl.jdbc.Schema=TRACE, console
log4j.category.com.solarmetric.kodo.Performance=TRACE, console
log4j.category.com.solarmetric.kodo.MetaData=TRACE, console
log4j.category.com.solarmetric.kodo.Enhance=TRACE, console
log4j.category.com.solarmetric.kodo.Query=TRACE, console
log4j.category.com.solarmetric.kodo.Runtime=TRACE, console

log4j.appender.console=org.apache.log4j.ConsoleAppender
```

**2.3.3. Logging using JDK 1.4 java.util.logging**

When using JDK 1.4 or higher, the built-in logging package provided by the [java.util.logging](#) package will be used. For details on configuring the built-in logging system, please see the [Java Logging Overview](#).

By default, JDK 1.4's logging package looks in the `JAVA_HOME/lib/logging.properties` file for logging configuration. This can be overridden with the `java.util.logging.config.file` system property. E.g., to run using a custom logging.properties file, the following command could be issued: `java -Djava.util.logging.config.file=mylogging.properties com.company.MyClass`

**Example 2.7. Example logging.properties file**

```
# Specify the handlers to create in the root logger
# (all loggers are children of the root logger)
# The following creates two handlers
handlers=java.util.logging.ConsoleHandler, java.util.logging.FileHandler

# Set the default logging level for the root logger
.level=ALL

# Set the default logging level for new ConsoleHandler instances
java.util.logging.ConsoleHandler.level=INFO

# Set the default logging level for new FileHandler instances
java.util.logging.FileHandler.level=ALL
```

[Prev](#)

[Next](#)

[2.2. Kodo JDO Properties](#)

[Home](#)

[Chapter 3. Creating Persistent Classes](#)

```
# Set the default formatter for new ConsoleHandler instances
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter

# Set the default logging level for all Kodo logs to INFO
com.solarmetric.kodo.impl.jdbc.SQL.level=INFO
com.solarmetric.kodo.impl.jdbc.JDBC.level=INFO
com.solarmetric.kodo.impl.jdbc.Schema.level=INFO
com.solarmetric.kodo.Performance.level=INFO
com.solarmetric.kodo.Metadata.level=INFO
com.solarmetric.kodo.Enhance.level=INFO
com.solarmetric.kodo.Query.level=INFO
com.solarmetric.kodo.Runtime.level=INFO
```

### 2.3.4. Logging using Simple Log

When a version of Java lower than 1.4 is being used, and Log4J libraries are not located in the CLASSPATH, then the commons logging package will fall back to using its built-in simple logging system, using the class org.apache.commons.logging.impl.SimpleLog. This system is controlled by the properties in the simplelog.properties resource, or else by various system properties, as specified at the [SimpleLog Javadoc](#).

**Example 2.8. Example simplelog.properties file**

```
# By default, we will log messages at "warn" or higher
org.apache.commons.logging.simplelog.defaultlog=warn

# formatting options
org.apache.commons.logging.simplelog.showShortLogname=true
org.apache.commons.logging.simplelog.showdatetime=true

# Set the default logging level for all Kodo logs to "info"
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.impl.jdbc.SQL=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.impl.jdbc.JDBC=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.impl.jdbc.Schema=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.Performance=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.Metadata=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.Enhance=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.Query=info
org.apache.commons.logging.simplelog.log.com.solarmetric.kodo.Runtime=info
```

### 2.3.5. Logging using a Custom Log

If it is ever the case where none of the built-in logging packages are suitable for an application, the logging system can be configured to use a custom logging class. This can be used, for example, to integrate with a proprietary logging framework used by some applications servers, or for logging to a graphical component for GUI applications.

Custom logging can be accomplished by writing an implementation of the [org.apache.commons.logging.LogFactory](#) class, and setting the org.apache.commons.logging.LogFactory system property to contain the name of the custom class.

[Prev](#)

2.2. Kodo JDO Properties

[Up](#)

[Home](#)

**Example 2.9. Example custom logging class**

Chapter 3. Creating Persistent Classes

```
public class CustomLoggingExample
    extends org.apache.commons.logging.impl.LogFactoryImpl
{
    public org.apache.commons.logging.Log getInstance (String name)
    {
        // return a simple extension of SimpleLog that will log
        // everything to the System.err stream.
        return new org.apache.commons.logging.impl.SimpleLog (name)
        {
            /**
             * In our example, all log levels are enabled.
             */
            protected boolean isLevelEnabled (int logLevel)
            {
                return true;
            }

            /**
             * Just send everything to System.err
             */
            protected void log (int type, Object message, Throwable t)
            {
                System.err.println ("CUSTOM_LOG: " + type + ": "
                    + message + ": " + t);
            }
        };
    }
}
```

---

[Prev](#)[2.2. Kodo JDO Properties](#)[Up](#)[Home](#)[Next](#)[Chapter 3. Creating Persistent Classes](#)