Table of Contents

Without log mechanisms, software development is a lot more arduous than it needs to be. For this reason, a log mechanism is included in JVx.

The following methods are provided via UIComponent:

```
public void info(Object... pInfo)
public void debug(Object... pInfo)
public void error(Object... pInfo)
```

Information (info) outputs are only relevant for the developer during the creation of the application. The debug outputs can also be helpful for the user. The display of error messages is helpful for troubleshooting and should be employed during productive use.

A simple method call is required to use logging during the development of applications, work screens or components. No special classes have to be used.

JVx Logging

JVx´s log mechanism does not define a new Logging API, although it does define an interface for the integration of logging frameworks.

The interface is defined via com.sibvisions.util.log.ILogger and used via com.sibvisions.util.log.LoggerFactory.

JVx contains an implementation for the Java Logging API!

To use JVx´s logging independently from the UI, the following steps are necessary:

Creating a logger:

```
ILogger log = LoggerFactory.getInstance(...);
```

Using the logger:

```
log.info(...);
```

JVx´s standard implementation limits the use of loggers. The actual instance of a logger (ILogger) is not initialized until it is accessed, i.e., a log output occurs.

The implementation of vararg parameters (Object...) positively affects performance and readability of the source code. The following constructs are not uncommon:

```
otherlogger.log("Row number: " + nr +" of " + count);
```

or

```
if (otherlogger.isLoggable(Level))
{
    otherlogger.log("Row number: " + nr +" of " + count);
}
```

In this case the first call always causes multiple string operations, even when the logger does not display a respective message. At the second call these string operations are avoided deliberately, although this results in more source code.

Using JVx, the log output would be coded as follows:

```
log.debug("Row number: ", nr, " of ", count);
```

We concede that an array operation is used here. However, this operation, unlike multiple string operations, does not significantly affect performance.

The logger´s configuration is dependent on the logging framework. For the Java Logging API, the file logging.properties can be used:

logging.properties

```
#######################################################################
#
# HANDLER definition
#######################################################################
#

# development handlers
handlers = java.util.logging.ConsoleHandler

# application handlers
#handlers = java.util.logging.FileHandler

#######################################################################
#
# HANDLER configuration
#######################################################################
#

java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter =
com.sibvisions.util.log.jdk.JdkLineFormatter

java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.pattern = application_%g.log
java.util.logging.FileHandler.limit = 10000
java.util.logging.FileHandler.count = 5
java.util.logging.FileHandler.formatter =
com.sibvisions.util.log.jdk.JdkLineFormatter

#######################################################################
#
# Package specifig log levels
#######################################################################
#
```

```
.level = OFF

#com.sibvisions.level = OFF
#com.sibvisions.rad.model.level = OFF
#com.sibvisions.rad.persist.level = ALL
#com.sibvisions.rad.server.level = OFF

#jvx.rad.level = ALL
```

The following methods can also be helpful for setting the log levels:

```
log.setLevel(level);

LoggerFactory.setLevel(name, level);
```

JVx´s log implementation can be used for RIAs, desktop, web and Android applications. For WebUI applications, it should be noted that the log outputs occur at the server.