

Oracle JDBC Logging using java.util.logging

An Oracle White Paper
June 2007

Oracle JDBC Logging using java.util.logging

[USE TABLE OF CONTENTS ONLY FOR TECHNICAL WHITE PAPERS. BUSINESS WHITE PAPERS SHOULD NOT BE SO LENGTHY THAT READERS NEED A CONTENTS PAGE.]

[After you have written the paper, follow these instructions to automatically generate the Table of Contents: Click in the Table of Contents below (this selects the entire table), then press the F9 key. Select 'Update Entire Table' and Word will automatically use your Level 1, 2, and 3 headings to generate the table. Remember to select the Table of Contents and then press F9, anytime you make an edit that might change the page numbers or heading text.]

Introduction.....	3
Configure the classpath.....	3
Enable Logging.....	3
Configure Logging.....	4
Advanced Configuration.....	5
Using Loggers.....	6
A Detailed Example.....	7
Conclusion.....	8

Oracle JDBC Logging using java.util.logging

INTRODUCTION

The Oracle JDBC drivers use two different mechanisms to generate log output. Versions of the JDBC drivers for older versions of Java, 1.2 and 1.3, use an proprietary mechanism. Versions of the JDBC drivers for newer versions of Java, 1.4 and later, use the Java standard logging mechanism, java.util.logging. This note describes how to use java.util.logging with the Oracle JDBC drivers.

Configure the classpath

Oracle ships several jar files for each version of the drivers. The optimized jar files do not contain any logging code. There will be no Oracle JDBC log output when using the optimized jar files. The jar files that do contain logging code are

- ojdbc5_g.jar
- ojdbc6_g.jar
- ojdbc5dms.jar *minimal logging*
- ojdbc6dms.jar *minimal logging*
- ojdbc5dms_g.jar
- ojdbc6dms_g.jar

Step 1: Make sure that a logging enabled jar file is the only Oracle JDBC jar file in your classpath.

Enable Logging

In order to get any log output from the Oracle JDBC drivers you must enable logging. There is a global switch that turns logging on and off. When it is off, the drivers will not produce any log output. When it is on, what logging is produced is controlled by the configuration of java.util.logging. There are two ways to enable the global logging switch, programmatically or setting a Java system property. You can use the programmatic way to control what parts of your program generate log output. If you cannot or do not want to change the source, you can set the Java system property to enable logging for the entire program execution.

Step 2a: globally enable logging by setting the oracle.jdbc.Trace system property

```
java -Doracle.jdbc.Trace=true ...
```

OR

Step 2b: In 11.1 you programmatically enable/disable logging with the following:

```
// compute the ObjectName
String loader
= Thread.currentThread().getContextClassLoader()
  .toString().replaceAll("[,=:\\"]+", "");
javax.management.ObjectName name
= new javax.management.ObjectName(
    "com.oracle.jdbc:type=diagnosability,name="+loader);

// get the MBean server
javax.management.MBeanServer mbs
= java.lang.management.ManagementFactory
  .getPlatformMBeanServer();

// find out if logging is enabled or not
System.out.println("LoggingEnabled = " +
    mbs.getAttribute(name, "LoggingEnabled"));

// enable logging
mbs.setAttribute(name,
    new javax.management.Attribute("LoggingEnabled", true));

// disable logging
mbs.setAttribute(name,
    new javax.management.Attribute("LoggingEnabled", false));
```

If this is all you do you will get minimal logging of serious errors written to the console. Usually this is less that useful. In order to generate more and probably more useful output, you must configure `java.util.logging`.

In 10g you programmatically enable/disable logging with the following:

```
oracle.jdbc.driver.OracleLog.setTrace(true); // enable logging
...
oracle.jdbc.driver.OracleLog.setTrace(false); // disable logging
```

If this is all you do you will get minimal logging of serious errors written to the console. Usually this is less that useful. In order to generate more and probably more useful output, you must configure `java.util.logging`.

Configure Logging

`java.util.logging` is a very rich and powerful tool. Describing all the things you can do with it is beyond the scope of this note. This note provides a basic set of tools that will let you generate useful log output. For more complex configurations look at the JavaDoc for `java.util.logging`.

You can configure `java.util.logging` either programatically or via a configuration file. For the most part there is little need to configure it programatically. You can turn Oracle JDBC logging on and off programmatically using `OracleLog.setTrace`.

In most cases there is no need to change the configuration during the course of execution. This note will only cover configuration files. If you must use programmatic configuration, information in the rest of this note should help you figure out what values to use when configuring `java.util.logging` programmatically.

One place to look for configuration information is the `OracleLog.properties` file in the demo directory of your JDBC installation. This file contains basic information on how to configure `java.util.logging` and provides some initial settings that you can start with. In order to use a config file you must identify that file to the Java runtime. You tell Java about your file by setting a system property. You can use both `java.util.logging.config.file` and `oracle.jdbc.Trace` at the same time.

Step 3: `java -Djava.util.logging.config.file=/jdbc/demo/OracleLog.properties -Doracle.jdbc.Trace=true ...`

This will use the default `OracleLog.properties` file (assuming it is reachable from `/jdbc/demo`). That may get you the output you want, or it may not. The rest of this note will show you how to create your own config file and in the process help you understand how to modify the sample `OracleLog.properties` file.

Step 4: create a file, for example `myConfig.properties`, and insert the following.

```
level=SEVERE
oracle.jdbc.level=FINE
oracle.jdbc.handlers=java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level=FINE
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
```

Save this file and execute the Java command above replacing `OracleLog.properties` with `myConfig.properties`. This can produce a large amount of output, so make sure your program execution is short.

Advanced Configuration

This gives us a working setup that traces everything to the console. Let's modify the config file to dump everything to a file instead. Instead of using the `ConsoleHandle`, use the `FileHandler`.

```
.level=OFF
oracle.jdbc.level=FINE
oracle.jdbc.handlers=java.util.logging.FileHandler
java.util.logging.FileHandler.level=FINE
java.util.logging.FileHandler.pattern = jdbc.log
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter
```

This will generate exactly the same log output, but instead send it to a file named `jdbc.log` in the current directory.

Step 5: You will want to reduce the amount of detail. You control the level of detail by changing the level settings. The defined levels from least detail to most are

```
OFF
SEVERE  SQLExceptions and internal errors
WARNING SQLWarnings and bad but not fatal internal conditions
INFO    Infrequent events
CONFIG  SQL strings
FINE    Calls to the public API
FINER   Calls to internal methods
FINEST  Calls to high volume internal methods
ALL
```

In order to reduce the amount of detail, change `java.util.logging.FileHandler.level` from `FINE` to `CONFIG`

```
java.util.logging.FileHandler.level=CONFIG
```

It is not necessary to change the level of the `oracle.jdbc` logger although you can. Setting the `FileHandler` level will control what log messages end up in the log file.

Using Loggers

Setting the level as above reduces all the logging output from JDBC. Sometimes we want to see a lot of output from one part of the code and very little from other parts. To do that you must understand more about loggers.

Loggers exist in a tree structure defined by their names. The root logger is named `""`, the empty String. If you look at the first line of the config file you see `.level=OFF`. This is setting the level of the root logger. The next line is `oracle.jdbc.level=FINE`. This sets the level of the logger named `oracle.jdbc`. The `oracle.jdbc` logger is a member of the logger tree. Its parent is named `oracle`. The parent of the `oracle` logger is the root logger. Logging messages are sent to a particular logger, for example `oracle.jdbc`. If the message passes the level check at that level the message is passed to the handler at that level, if any, and to the parent logger. So a log message sent to `oracle.log` is compared against that logger's level, `CONFIG` if you are following along. If the level is the same or less (less detailed) then it is sent to the `FileHandler` and to the parent logger, `oracle`. Again it is checked against the level. If as in this case, the level is not set then it uses the parent level, `OFF`. If the message level is the same or less it is passed to the handler, which there isn't one, and sent to the parent. In this case the parent is the root logger.

All this tree stuff didn't help you reduce the amount of output. What will help is that the JDBC drivers use several subloggers. If you restrict the log messages to one of the subloggers you will get substantially less output. The loggers used by the Oracle JDBC drivers include

<code>oracle.jdbc</code>	almost all Oracle JDBC messages
<code>oracle.jdbc.driver</code>	the core driver code
<code>oracle.jdbc.pool</code>	DataSources and Connection pooling
<code>oracle.jdbc.rowset</code>	RowSets
<code>oracle.jdbc.xa</code>	distributed transaction support
<code>oracle.sql</code>	complex SQL data types

The drivers may use other loggers as well. That will vary from release to release.

A Detailed Example

Suppose you want to trace what is happening in the `oracle.sql` component, but you also want to capture some basic information about the rest of the driver. This is a more complex use of logging. Here is the config file.

```
#
# set levels
#
1 level=SEVERE
2 oracle.level=ALL
3 oracle.jdbc.driver.level=INFO
4 oracle.jdbc.pool.level=OFF
5 oracle.jdbc.util.level=OFF
6 oracle.sql.level=FINE
#
# Config handlers
#
7 oracle.handlers=java.util.logging.ConsoleHandler
8 java.util.logging.ConsoleHandler.level=ALL
9 java.util.logging.ConsoleHandler.formatter = \
  java.util.logging.SimpleFormatter
```

Let's consider what each line is doing.

- 1 Set the root logger to `SEVERE`. We don't want to see any logging from other, non-Oracle components unless something fails badly, so we set the default level for all loggers to `SEVERE`. Each logger inherits its level from its parent unless set explicitly. By setting the root logger to `SEVERE` we insure that all other loggers inherit that level except for the ones we set otherwise.

- 2 We want output from both the oracle.sql and oracle.jdbc.driver loggers. Their common ancestor is oracle, so we set the level there to ALL. We will control the detail more explicitly at lower levels.
- 3 We only want to see the SQL execution from oracle.jdbc.driver so we set that to INFO. This is a fairly low volume level but will allow us to keep track of what our test is doing.
- 4 We are using a DataSource in our test and don't want to see all of that logging so we turn it OFF.
- 5 Similarly we don't want to see the logging from the oracle.jdbc.util package. If we were using XA or rowsets we would turn them off as well.
- 6 We want to see how our app is using oracle.sql so we set oracle.sql.level to FINE. This provides a lot of information about the public method calls without overwhelming detail.
- 7 We are going to dump everything to stderr. When we run the test we will redirect stderr to a file.
- 8 We want to send everything to the console. We are doing the filtering with the loggers rather than the Handler this time.
- 9 We will use a simple, more or less human readable format. Another choice is XMLFormatter. XMLFormatter is the best choice for logs that you send to Oracle Support because it permits us to more easily use automated processing of the log output.

When you run your test with this config file you will get an XML document that contains moderately detailed information from the oracle.sql package, a little bit of information from the core driver code and nothing from any other code.

CONCLUSION

This paper shows you all the tools you will need for most purposes. java.util.logging is a powerful tool with lots of switches and knobs. You can send different parts of the log stream to different places and write custom filters that pick out exactly the log messages you want to see. These advanced uses are beyond the scope of this note. The basic tools described above should cover most of your needs and will give you a head start in learning the more advanced techniques. The best source for more information about java.util.logging is the JavaDoc.

One final note. The Oracle JDBC logging code varies dramatically from release to release. We are constantly striving to make it more useful and more maintainable. Older releases don't always do exactly what we or you would like them to do. This note more describes a philosophy rather than exactly what will happen in any given release. The techniques described here should get you something close to what you want, even if not exactly. Experiment. This note describes how we want logging to work and each subsequent release should be closer to this ideal. I hope this helps.



Oracle JDBC Logging Using java.util.logging
August 2007

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2007, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.