

# Platforms

by Thomas Mahler

## Table of contents

1	How to use OJB with a specific relational database.....	2
2	Basic Concepts.....	2
2.1	OJB internal tables.....	2
2.2	Tables for the regression testbed.....	4
2.3	Tables for the tutorial applications.....	4
3	The setup process.....	4
3.1	Selecting a platform profile.....	4
3.2	editing the profile to point to your target db.....	5
3.3	Executing the build script.....	5
3.4	Verifying the installation.....	6

## 1. How to use OJB with a specific relational database

OJB has been designed to smoothly integrate with any relational database that provides JDBC support. OJB can be configured to use only JDBC 1.0 API calls to avoid problems with restrictions of several JDBC drivers.

It uses a limited SQL subset to avoid problems with restrictions of certain RDBMS. This design allows to keep the OJB code generic and free from database specifics.

This document explains basic concepts and shows how OJB can be configured to run against a specific RDBMS.

If you not already have done so, then you also might want to have a look at the [Getting Started](#) section which presents a sample skeleton project.

## 2. Basic Concepts

### 2.1. OJB internal tables

For certain features OJB relies on several internal tables that must be present in the target RDBMS to allow a proper functioning. The associated internal object [metadata mapping](#) of these internal used tables can be found in [repository\\_internal.xml](#) file.

If those features are not needed/used OJB can be safely run without any internal tables and metadata mapping.

The following table lists all tables and their specific purpose.

Table	Purpose
<p>OJB_HL_SEQ</p> <pre>&lt;table name="OJB_HL_SEQ" description="HIGH/LOW SequenceManager table"&gt;   &lt;column name="TABLENAME" required="true" primaryKey="true" type="VARCHAR" size="250"/&gt;   &lt;column name="MAX_KEY" type="BIGINT"/&gt;   &lt;column name="GRAB_SIZE" type="INTEGER"/&gt;   &lt;column name="VERSION" type="INTEGER"/&gt; &lt;/table&gt;</pre>	<p>Table for the high/low sequence manager. The column <i>TABLENAME</i> was used to persist the sequence name (may be re-named in further versions of OJB). If the built-in OJB sequence manager is not used, this table is not needed.</p>
<p>OJB_NRM</p> <pre>&lt;table name="OJB_NRM" description="OJB NAMED ROOTS Table"&gt;   &lt;column name="NAME" required="true" primaryKey="true" type="VARCHAR" size="250"/&gt;   &lt;column name="OID_" type="LONGVARBINARY"/&gt; &lt;/table&gt;</pre>	<p>The "Named Roots Map". ODMG allows to bind persistent objects to an user defined name - called <a href="#">named objects</a>. The Named roots map is used to store these bindings. It has NAME (String of arbitrary length) as primary key and keeps the serialized OID of a persistent object or an arbitrary serialized object in the field OID (String of arbitrary length). If <code>Database.bind(...)</code> and <code>Database.lookup(...)</code> are not used in client apps, this table is not needed.</p>
<p>OJB_DLIST</p> <pre>&lt;table name="OJB_DLIST" description="DLIST IMPLEMENTATION"&gt;   &lt;column name="ID"</pre>	<p>The table used for the ODMG persistent DList collections. If ODMG DLists are not used, this table is not needed.</p>

<pre>required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="SIZE_" type="INTEGER"/&gt; &lt;/table&gt;</pre>	
<p>OJB_DLIST_ENTRIES</p> <pre>&lt;table name="OJB_DLIST_ENTRIES" description="DList entry table"&gt;     &lt;column name="ID" required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="DLIST_ID" required="true" type="INTEGER"/&gt;     &lt;column name="POSITION_" type="INTEGER"/&gt;     &lt;column name="OID_" type="LONGVARBINARY"/&gt; &lt;/table&gt;</pre>	<p>stores the entries of DLists (a wrapper to objects stored in the DList)</p> <p>If ODMG DLists are not used, this table is not needed.</p>
<p>OJB_DSET</p> <pre>&lt;table name="OJB_DSET" description="DSET IMPLEMENTATION"&gt;     &lt;column name="ID" required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="SIZE_" type="INTEGER"/&gt; &lt;/table&gt;</pre>	<p>The table used to store ODMG persistent DSET collections</p> <p>If ODMG DSets are not used, this table is not needed.</p>
<p>OJB_DSET_ENTRIES</p> <pre>&lt;table name="OJB_DSET_ENTRIES" description="DSet entry table"&gt;     &lt;column name="ID" required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="DLIST_ID" required="true" type="INTEGER"/&gt;     &lt;column name="POSITION_" type="INTEGER"/&gt;     &lt;column name="OID_" type="LONGVARBINARY"/&gt; &lt;/table&gt;</pre>	<p>This table stores the entries of DSets.</p> <p>If ODMG DSets are not used, this table is not needed.</p>
<p>OJB_DMAP</p> <pre>&lt;table name="OJB_DMAP" description="DMap table"&gt;     &lt;column name="ID" required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="SIZE_" type="INTEGER"/&gt; &lt;/table&gt;</pre>	<p>The table use to store the ODMG persistent DMap tables</p> <p>If ODMG DMaps are not used, this table is not needed.</p>
<p>OJB_DMAP_ENTRIES</p> <pre>&lt;table name="OJB_DMAP_ENTRIES" description="DMap entry table"&gt;     &lt;column name="ID" required="true" primaryKey="true"     type="INTEGER"/&gt;     &lt;column name="DMAP_ID" required="true" type="INTEGER"/&gt;     &lt;column name="KEY_OID" type="VARBINARY"/&gt;     &lt;column name="VALUE_OID" type="VARBINARY"/&gt;</pre>	<p>The table containing the DMap entries. The Keys and Values of the map can be arbitrary persistent objects.</p> <p>If ODMG DMaps are not used, this table is not needed.</p>

</table>	
----------	--

OJB uses [Torque](#) to create all required tables and data. Thus there is no SQL DDL file, but an XML file describing the tables in format readable by Torque. The Torque DDL information for the internal tables resides in the file `src/schema/ojbcore-schema.xml`.

The o/r mappings for these tables are contained in the file `repository_internal.xml`.

If you want to have a look at how these files could be used, have a look at the the [ojb-blank](#) sample project which is already prepared to use these files.

## 2.2. Tables for the regression testbed

It is recommended to run the [OJB test-suite](#) against your target database. Thus you will have to provide several more tables, filled with the proper testdata.

The DDL information for these tables resides in the file `src/schema/ojbtest-schema.xml`.

The testdata is defined in the file `src/schema/ojbtest-data.xml`.

The o/r mappings for these tables are contained in the file `repository_junit.xml`.

## 2.3. Tables for the tutorial applications

If you intend to run the OJB tutorial applications against your target database you will have to provide one extra table.

The DDL information for this table also resides in the file `src/schema/ojbtest-schema.xml`.

The testdata is also defined in the file `src/schema/ojbtest-data.xml`.

The o/r mappings for this table is contained in the file `repository_user.xml`.

## 3. The setup process

OJB provides a setup routine to generate the target database and to fill it with the required testdata. This routine is based on Torque scripts and is driven from the `build.xml` file. This section describes how to use it.

### 3.1. Selecting a platform profile

OJB ships with support for several popular database platforms. The target platform is selected by the switch `profile` in the file `build.properties`. You can choose one out of the predefined profiles:

```
# With the 'profile' property you can choose the RDBMS platform OJB is using
# implemented profiles:
#
profile=hsqldb
# use the mssqldb-JSQLConnect profile for Microsoft SQL Server and
# you will automatically JSQLConnect driver, from http://www.j-netdirect.com/
# MBAIRD: This is my driver of preference for MS SQL Server, I find the OEM'd
# MS driver to have some problems.
#profile=mssqldb-JSQLConnect
#profile=mssqldb-Opta2000
#profile=mssqldb-ms
#profile=mysql
#profile=db2
#profile=oracle
```

```
#profile=oracle9i
#profile=oracle9i-Seropto
#profile=msaccess
#profile=postgresql
#profile=informix
#profile=sybase
#profile=sapdb
#profile=maxdb
```

The profile switch activated in `build.properties` is used to select a profile file from the profile directory.

If you set `profile=db2`, then the file `profile/db2.profile` is selected.

This file is used by the Torque scripts to set platform specific properties and to perform platform specific SQL operations.

### 3.2. editing the profile to point to your target db

The platform specific file `profile/xxx.profile` contains lots of information used by Torque. You can ignore most of it. The only important part in this file is the section where the url to the target db is assembled, here is an snip of the DB2 profile:

```
# -----
#
#  D A T A B A S E  S E T T I N G S
#
# -----
# JDBC connection settings. This is used by the JDBCtoXML task
# that will create an XML database schema from JDBC metadata.
# These settings are also used by the SQL Ant task to initialize
# your Turbine system with the generated SQL.
# -----

dbmsName = Db2
jdbcLevel = 1.0
urlProtocol = jdbc
urlSubprotocol = db2
urlDbalias = OJB

createDatabaseUrl = ${urlProtocol}:${urlSubprotocol}:${urlDbalias}
buildDatabaseUrl = ${urlProtocol}:${urlSubprotocol}:${urlDbalias}
databaseUrl = ${urlProtocol}:${urlSubprotocol}:${urlDbalias}
databaseDriver = COM.ibm.db2.jdbc.app.DB2Driver
databaseUser = admin
databasePassword = db2
databaseHost = 127.0.0.1
```

These settings result in a database URL `jdbc:db2:OJB`. If your production database is registered with the name `MY_PRODUCTION_DB` you have to edit the entry `urlDbalias` to:

```
urlDbalias = MY_PRODUCTION_DB.
```

In this section you can also set application user name and password. You can also enter a different jdbc driver class, to activate a different driver.

Before progressing, please check that the jdbc driver class, named in the `databaseDriver` entry is located on the classpath! You can either edit the global environment variable `CLASSPATH` or place the jdbc driver jar file into the `jakarta-objb-xxx/lib` directory.

### 3.3. Executing the build script

Now everything should be prepared to launch the setup routine. This routine can be invoked by calling **ant prepare-testdb**.

If you are prompted with a `BUILD SUCCESSFUL` message after some time, everything is OK.

If you are prompted with a `BUILD FAILED` message after some time, something went wrong. This may have several reasons:

- You entered some incorrect settings. Please check the log messages to see what went wrong.
- Torque does not work properly against your target database. Torque is very flexible and should be able to work against a wide range of databases. But the code templates for each database may not be accurate. Please check the `obj-user` mailinglist archive if there are any failure reports for your specific database. Please also check if someone contributed a fix already. If you don't find anything please post your problem to the `obj-user-list`.

As a last resort you can try the following: Switch back to the default `hsqldb` profile and execute `ant prepare-testdb`. This will setup the default `hsqldb` database. And it will also generate SQL scripts that you may use to generate your database manually.

The SQL scripts are generated to `jakarta-obj-xxx/target/src/sql`. You can touch these scripts to match your database specifics and execute them manually against your platform.

### 3.4. Verifying the installation

Now everything is setup to run the junit regression tests against your target database.

Execute

```
ant junit
```

to see if everything works as expected. more information about the [OJB Test Suite here](#). If you did not manage to set up the target database with the `ant prepare-testdb` you can use `ant junit-no-compile-no-prepare` to run the testsuite without generation of the test database.