TopicMap Engine Home

***TM2JDBC** is a implementation of the TopicMaps API (TMAPI 2.0.2). It is a fast, production-grade Topic Maps Engine with a small dependency footprint. TM2JDBC uses a relational database to store its data, and is verified to operate with Apache Derby, MySQL, MSSQL Server and Oracle databases.*

TM2JDBC is released in version 1.0.1. This release features some dependency updates and minor corrections of inconsistencies within the code, but is fully compatible to version 0.9.0.

- Updated TMAPI dependency to version 2.0.2 (the prior version already was compliant with TMAPI 2.0.2, but still referenced a snapshot as the final library was not released)
- OSGi support – The library now comes packaged as an OSGi-bundle, which allows it to be deployed to your OSGi container of choice. This has no side-effencts on non-OSGi projects
- Updated clazzes.org-dependencies to the current version

If you are using maven, you can use the new version by adding the following snippets to your `pom.xml` file:

```
<dependency>
  <groupId>org.clazzes</groupId>
  <artifactId>tm2jdbc</artifactId>
  <version>1.0.1</version>
</dependency>


...

<repository>
  <id>org.clazzes</id>
  <name>Clazzes.org repository.</name>
  <url>http://maven.clazzes.org</url>
</repository>
```

If you do not use maven, you can download the .jar files manually from our repository at http://maven.clazzes.org/org/clazzes/tm2jdbc.
Alternatively, you can check out the sources directly from our SVN Repository:

```
http://svn.clazzes.org/svn/topicmaps/trunk/tm2jdbc/
```

Setup and Usage Notes

TM2JDBC needs a relational database and the appropriate JDBC 4.0 compatible driver to function. However it does not have any direct dependencies to a specific JDBC connector. Instead, it expects a `javax.sql.DataSource` to be set as a property (see below), which it will use to obtain database connections. Although the engine should function with any DataSource implementation, we strongly recommend to use the Apache Commons DBCP library . A rudimentary set-up for a `BasicDataSource` can be found in the class `org.clazzes.tm2jdbc.util.test.DataSourceProvider`.

Providing a DataSource

To handle database connectivity, the TM2JDBC engine relies on `ajavax.sql.DataSource` to be set for the property `http://psi.clazzes.org/topicmaps/db-connection/datatsource` in the Factory (attempting to call `newTopicMapSystem()` without the property being set causes an`IllegalStateException` to be thrown. The property string can also be accessed directly via the Enum object`org.clazzes.tm2jdbc.voc.PropertyKey.CONNECTION_DATASOURCE`. The most basic setup could look like this:

```
BasicDataSource datasource = new BasicDataSource();
datasource.setDefaultAutoCommit(false);

datasource.setInitialSize(4);
datasource.setMinIdle(1);
datasource.setMaxIdle(8);
datasource.setMaxActive(8);
datasource.setMaxWait(3000);

datasource.setDriverClassName(driver);

datasource.setPassword(password);
datasource.setUsername(user);
datasource.setUrl(url);

TopicMapSystemFactory factory = TopicMapSystemFactory.newInstance();
factory.setProperty(PropertyKey.CONNECTION_DATASOURCE.getURI(), dsProvider.getDataSource());

TopicMapSystem system = factory.newTopicMapSystem();
```

Running the DBSetup Utility in commandline

The engine provides a setup-utility to create the tables it needs in a provided database schema. This utility can either be run as a standalone java executable (in the command line, via a shell script etc.) or instanciated as a Java Object by your own code (see below). The class name is `org.clazzes.tm2jdbc.jdbc.setup.DBSetup`. For testing and development, it may be useful to run the setup utility in commandline or the development evironment of your choice. The standard use-case is to set up the database schema, but you can also use it to drop all tables in the schema.
Making sure that the appropriate database drivers are added to the classpath, the accepts arguments like this:

```
DBSetup {derby|mssql|mysql|oracle} {{ [-h <HOSTNAME>] [-t <PORT>] [-n <SCHEMA NAME>] } | {--url <DATABASE-URL>}} -u <USER> -p <PASSWORD> [-d <DRIVER-CLASSNAME> ] [--debug] [--drop|--force-drop] [--nosetup]
```

Below is a listing of all argument flags and the default values:

Flag	Default Value	Description
`-h`	127.0.0.1	The hostname/address of your database server. **Caution: When using Oracle database systems, the hostname is interpreted as a TNS name, which is resolved to an IP locally. Alternatively, you may also specify a string like`<IP-HOST>/<SID>`**
`-t`	*default port*	If omitted, the default port for the specified database server is used
`-n`	topicmaps	
`--url`	*none*	Alternatively to entering the hostname, port and schema name, you can supply the jdbc url directly.
`-u`	*none*	You must enter a username and password
`-p`	*none*	
`-d`	*default driver class*	If you want to use a different driver class than the default, you can specify the fully qualified classname using this flag. Make sure that the class is added to the classpath.
`--debug`	*false*	Produces a verbose output
`--drop`	*false*	Drops all tables in the schema, causes the utility to fail on errors
`--force-drop`	*false*	Drops all tables in the schema, ignoring any errors (Use to clean the schema if it was corrupted or not created completely)

`--nosetup`	*false*	Skip setting up the schema.

Running the DBSetup Utility in Java

The database setup-utility also can be run in java. For this case it provides a constructor which takes a `javax.sql.DataSource`, and provides the methods `setUpDB(boolean debug)` and `dropAllTables(boolean supressErrors)`. It is recommended to use these instead of running the `main (String[] args)` method with the appropriate arguments.

JUnit testing

TMAPI provides an extensive suite of Testcases to assure compliancy, which is also included in the TM2JDBC package. As the original tests were not conceived for engines which are based on an RDBMS, the `setUp()` and `tearDown()` methods where modified to initialize and drop the database schema between every test. You are welcome to subject the engine to additional tests (we would especially appreciate you to share the results with us), and for this purpose TM2JDBC provides the utility class`org.clazzes.tm2jdbc.util.test.DataSourceProvider` (depends on Apache commons DBCP).
It is recommended to use this utility in all test cases as follows:

```
protected DataSourceProvider dsProvider;

protected void setUp() throws Exception {
        super.setUp();
        if (dsProvider == null) {
                dsProvider = DataSourceProvider.getProvider();
                dsProvider.setUpDB();

                //set up factory etc. here ...
                factory.setProperty(PropertyKey.CONNECTION_DATASOURCE.getURI(), dsProvider.getDataSource());
                //...
        }
}

protected void tearDown() throws Exception {
        dsProvider.dropDB();
}
```

Source code and issue tracker

Subversion: http://svn.clazzes.org/svn/topicmaps

Jira: TME