How to add support for new databases

1 Overview

If you want to add support for an unsupported database system, you found the right documentation. This step-by-step guide describes all the steps that need to be processed in order to support a new database system.

1.1 QA Test suite

First we need to make sure, what is the qa test suite. With Activiti there are two projects that make up the qa. These are:

1. activiti-engine-examples
2. activiti-engine-test-api

These tests need to be run successfully before we consider Activiti to support a database system.

1.2 Run a test with H2

Before we start to go through all the steps, we should execute the test scripts on an already supported database. We assume, you already checked out the activiti project, setup the demo and configured ant and maven on your system.

Before we can run the test, we need to provide a database connection. This is done via a properties file located in <user_home>/.activiti/jdbc. There create a file called h2.properties adding this content:

h2.properties

```
jdbc.driver=org.h2.Driver
jdbc.url=jdbc:h2:tcp://localhost/activiti
jdbc.username=sa
jdbc.password=
```

This setup assumes an installed h2 database with no tables in it! All tests are constructed to create the database before running the test and dropping the database after finishing the tests. If you have setup the database and the configuration file, you can proceed with the next command.

Navigate to the “activiti-engine-examples”-project and execute following command:

mvn --Ddatabase=h2 clean install

You should see lots of info on the console. Most important are the tests that are run with this command. You also saw the creation and dropping of the database before and after the testing. This defines another point in the configuration for a new database system.

2 Step-by-Step Guide

Now we will describe the parts that have to be completed for the purpose of this document. These are:

1. Setup a database with schema and user
2. Modify the database creation and deletion
3. Add configuration file to activiti´s user_home
4. Modify the pom.xml
5. Running the test project
6. Analyze test-results
7. Run QA Test suite script

2.1 Setup a database with schema and user

This should be the easiest part of the description. You need to setup the database you want to test with a schema and a user. The user needs to have enough privileges to execute all the following sql statements.

You may already write down the information that is needed to connect to this database and fetch the jdbc driver for this database. Then you can proceed to the next step.

2.2 Modify the database creation and deletion

Activiti uses two sql files to create and drop the database tables. These are located at:

/activiti-engine/src/main/resources/org/activiti/db/create/activiti.<db-profile>.create.sql

/activiti-engine/src/main/resources/org/activiti/db/drop/activiti.<db-profile>.drop.sql

<db-profile> is an identifier; you would also use it to run your tests against. We did that in chapter “Run a test with H2”. Before you can start the testing of Activiti against a new database, you need to setup these two scripts. Start with copying the two files and rename them to your <db-profile> setting. After that you need to work out create, alter and drop statements for your database system. Be cautious to place those files in the same directory!

2.3 Add configuration file to activiti´s user home

As we did in the introduction sample, you have to provide the database connection configuration in the activiti user home directory. Please create a properties file like this:

<db-profile>.properties

You need to add the jdbc parameters like this

<db-profile>.properties

```
jdbc.driver=org.h2.Driver
jdbc.url=jdbc:h2:tcp://localhost/activiti
jdbc.username=sa
jdbc.password=
```

This is just a sample configuration for h2! You need to change the properties to your database environment.

2.4 Modify the pom.xml

Since the database connection is achieved by jdbc, you need to provide a jdbc-driver for the tests. There are two ways you can provide the jdbc driver:

1. Adding the appropriate dependency to maven pom.xml
2. Install jdbc driver to your local maven repository

If you have a maven dependency xml snippet for your jdbc driver, fine! Then you can skip the next step and go directly to step 2 “Modify maven pom.xml”. Otherwise you need to install the library into your local maven repository first.

1 Install the library into your local maven repository

mvn install:install-file -Dfile=<path to library> -DgroupId=<groupId>-DartifactId=<artifactId> -Dversion=<version> -Dpackaging=jar

Example for mysql:

mvn install:install-file -Dfile=mysql.jar -DgroupId=mysql -DartifactId=mysql-connector-java driver -Dversion=5.1.6 -Dpackaging=jar

After having installed the library you should now get the dependency snippet like this:

```
<dependency>
   <groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId
>
   <version>5.1.6</version>
   *<scope>test</scope>*
</dependency>
```

You can this example as basis and add your dependency information here. Make sure you have the scope property set to “test”. This dependency is only needed for tests. In a running Activiti scenario you would provide the jdbc driver for the activiti-engine. But this goes beyond the scope of this document.

2 Modify the pom.xml

If you open the pom.xml from one of the mentioned projects, you will see an xml file. Navigate to the dependencies area of the database profile. This should like this:

```
<profiles>
   <profile>
      <id>database</id>
      <activation>
         <property>
            <name>database</name>
         </property>
      </activation>
      <dependencies>
         <dependency>
            <groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId
>
            <version>5.1.6</version>
            <scope>test</scope>
         </dependency>
         <ADD YOUR DEPENDENY HERE>
      </dependencies>
      <build>
```

You can add your database driver dependency at the marked location. Save the file and proceed to the next step. You may return to this step to also modify the pom.xml for the other activiti project.

2.5 Runing the test project

You can run the test from the command line located in the appropriate test project. This is either “activiti-engine-examples” or “activiti-engine-test-api”. To be sure that your changes in project activiti-engine are up to date, run

mvn clean install

from the activiti-engine project location. After that execute this command:

mvn --Ddatabase=<db-profile> clean install

* *If you got lucky and the test didn´t fail one test, you can proceed with the other project. For the unlucky guys like mine, proceed with the next step!

2.6 Analyze test results

This is the hardest part and I can only give you some advice how to get to the cause of test failures. Any test logs errors and output to the target folder. In case of errors search for

/target/surefire-reports

These reports should help you a lot to get the conditions on which those errors rely on. The following points are at least a checklist or advice list I found out adding support for some databases:

- If tests fail, drop the database table yourself before running the mvn command once again. The database will only be dropped, if the tests fail on assertion, not on exceptions.
- Debugging tests might help to get the processing right. This helped me a lot to encounter wrong myBatis sqls.
- If you get errors like “database not clean” or “expected 0 got 3”, this could be due to timing problems. We had lots of them adding mysql! But these might also derive from system performance. At least for some tests with timers.
- Sorting algorithm might also be a point you should consider.
- Always start investigating on the first job that failed. The others might be due to the circumstances that flawed test consigned.

If you encounter a bug in the test, you can file a bug in JIRA! Otherwise after you fixed something you can once again run the test.

2.7 Run QA test suite script

Last step is to successfully run the qa script. This test combines creation of activiti-engine and executing all tests for database support. If this test is successful, you have added support for your database environment. We would be happy, if you would provide those scripts to the community.

2.7.1 Setup / Configuration

The following lines will guide you through running the test suite. For this example we will use MySQL 5.1 as database environment.

- **Checkout activiti project from SVN**
  Before you can run the test suite, you need to [checkout activiti|../display/ACT/Developers+Guide#DevelopersGuide-Buildingadistribution\|]. The qa tests are located in the projects 'activiti-engine-examples' and 'activiti-engine-test-api', but you do not need to start them from there separately. You will start the tests from the quality assurance project. But before we have to create a new database schema and user.
- **Create a new schema and user**
  First we need to create a new schema and a new user for the database connection. We set the schema name to 'activiti'.The users name and password are 'qa'! If you need guidance on how to create a new schema and user for MySQL, please consult their documentation. Now we need to make sure that the test suite gets a database connection.
- **Create a database connection file**
  You need to provide databse connection properties for the test suite to be executed. This information goes

into a file called 'mysql.properties'. Please create the following folder structure:
/
Unknown macro: {user_home}
/.activiti/jdbc
Within this directoy create a new file mysql.properties like this:jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://<hostname>:<port>/activiti
jdbc.username=qa
jdbc.password=qa

2.7.2 Run the test suite

Now we are ready to run the qa script. Navigate to /activiti/trunk/qa/ci and run the script that is suitable for your operating system like this:run-test-db-standalone mysqlIf you would like to run the qa script against H2, then you need to execute this command:run-test-db-standalone h2