Living Documentation

Version 1.0.0-SNAPSHOT

Table of Contents

Chapter 1. Introduction

DBUnit Rules aims for bringing DBUnit closer to your JUnit tests. Here are the main features:

- JUnit rule to integrate with DBUnit via annotations:

```
    @Rule
    public DBUnitRule dbUnitRule = DBUnitRule.instance(jdbcConnection);①

    @Test
    @DataSet(value = "datasets/yml/users.yml")
    public void shouldSeedDataSet(){
        //database is seed with users.yml dataset
    }
```

① The rule depends on a JDBC connection.

- CDI interceptor to seed database without rule instantiation;
- Json, Yaml, xml and flat xml support;
- Cucumber integration;
- JPA integration;
- Multiple database support;
- Date/time support in datasets;

The project is composed by 5 modules:

- Core: Contains the dataset executor and JUnit rule;
- CDI: provides the DBUnit interceptor
- Cucumber: a CDI aware cucumber runner;
- JPA: Comes with a dataset executor based on JPA entity manager
- Examples module.

Chapter 2. Summary

Scenarios			Steps							Features: 5	
Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Total	Duration	Status
Manage database with DBUnit Rules Core											
1	0	1	4	0	0	0	0	0	4	017ms	passed
Manage database with DBUnit Rules CDI											
1	0	1	4	0	0	0	0	0	4	06s 680ms	passed
Manage database with DBUnit Rules Cucumber											
1	0	1	5	0	0	0	0	0	5	006ms	passed
Dynamic data using scritable datasets											
2	0	2	7	0	0	0	0	0	7	011ms	passed
Database assertion using expected datasets											
5	0	5	16	0	0	0	0	0	16	021ms	passed
Totals											
10	0	10	36	0	0	0	0	0	36	06s 737ms	

Chapter 3. Features

3.1. Manage database with DBUnit Rules Core

> In order to manage database state in JUnit tests
> As a developer
> I want to use DBUnit in my tests.

DBUnit Rules Core module brings DBunit to your unit tests via JUnit rules.

3.1.1. Scenario: Seed database using yml dataset

Given

The following junit rules 👍 (016ms)

```
@RunWith(JUnit4.class)
public class DBUnitRulesIt {
    @Rule
    public EntityManagerProvider emProvider =
EntityManagerProvider.instance("rules-it"); ①

    @Rule
    public DBUnitRule dbUnitRule =
DBUnitRule.instance(emProvider.connection()); ②
}
```

① EntityManagerProvider is a simple Junit rule that creates a JPA entityManager for each test. DBunit rule don't depend on EntityManagerProvider, it only needs a JDBC connection.

② DBUnit rule responsible for reading `@DataSet` annotation and prepare the database for each test.

And

The following dataset 👍 (000ms)

src/test/resources/dataset/yml/users.yml

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
tweet:
  - id: abcdef12345
    content: "dbunit rules!"
    user_id: 1
  - id: abcdef12233
    content: "dbunit rules!"
    user_id: 2
  - id: abcdef1343
    content: "CDI for the win!"
    user_id: 2
follower:
  - id: 1
    user_id: 1
    follower_id: 2
```

When

The following test is executed: 👍 (000ms)

```
    @Test
    @DataSet(value = "datasets/yml/users.yml", useSequenceFiltering =
true)
    public void shouldSeedUserDataSet() {
        User user = (User) em().createQuery("select u from User u join
fetch u.tweets join fetch u.followers where u.id =
1").getSingleResult();
        assertThat(user).isNotNull();
        assertThat(user.getId()).isEqualTo(1);
        assertThat(user.getTweets()).isNotNull().hasSize(1);
        Tweet tweet = user.getTweets().get(0);
        assertThat(tweet).isNotNull();
        Calendar date = tweet.getDate();
        Calendar now = Calendar.getInstance();

assertThat(date.get(Calendar.DAY_OF_MONTH)).isEqualTo(now.get(Calendar.
DAY_OF_MONTH));
    }
```

Then

The database should be seeded with the dataset content before test execution 👍 (000ms)

3.2. Manage database with DBUnit Rules CDI

In order to manage database state in **CDI** based tests
As a developer
I want to use DBUnit in a CDI test environment.

DBUnit CDI integration is done through a CDI interceptor.

CDI must be enabled in your test, see the following example:

```
@RunWith(CdiTestRunner.class) ①
public class DBUnitCDITest {

}
```

① CdiTestRunner is provided by Apache Deltaspike but you should be able to use other CDI test runners.

3.2.1. Scenario: Seed database using yml dataset

Given

DBUnit interceptor is enabled in your test beans.xml: 👍 (06s 680ms)

src/test/resources/META-INF/beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
     <interceptors>

<class>com.github.dbunit.rules.cdi.DBUnitInterceptor</class>
     </interceptors>
</beans>
```

Your test itself must be a CDI bean to be intercepted. if you're using Deltaspike test control just enable the following property in `test/resources/META-INF/apache-deltaspike.properties`:

```
deltaspike.testcontrol.use_test_class_as_cdi_bean=true
```

And

The following dataset 👍 (000ms)

src/test/resources/dataset/yml/users.yml

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
tweet:
  - id: abcdef12345
    content: "dbunit rules!"
    user_id: 1
  - id: abcdef12233
    content: "dbunit rules!"
    user_id: 2
  - id: abcdef1343
    content: "CDI for the win!"
    user_id: 2
follower:
  - id: 1
    user_id: 1
    follower_id: 2
```

When

The following test is executed: 👍 (000ms)

```
Unresolved directive in documentation.adoc -
include::../../src/test/java/com/github/dbunit/rules/DBUnitCDITest.java
[tags=seedDatabase]
```

Then

The database should be seeded with the dataset content before test execution 👍 (000ms)

3.3. Manage database with DBUnit Rules Cucumber

> In order to manage database state in BDD tests
> As a BDD developer
> I want to use DBUnit along side my BDD tests.

DBUnit enters the BDD world through a dedicated JUNit runner which is based on Cucumber and Apache DeltaSpike.

This runner just starts CDI within your BDD tests so you just have to use DBUnit rules CDI interceptor on Cucumber steps, here is the so called Cucumber CDI runner declaration:

```
package com.github.dbunit.rules.bdd;

import com.github.dbunit.rules.cucumber.CdiCucumberTestRunner;
import cucumber.api.CucumberOptions;
import org.junit.runner.RunWith;

/**
 * Created by rmpestano on 4/17/16.
 */
@RunWith(CdiCucumberTestRunner.class)
@CucumberOptions(features = {
        "src/test/resources/features/core/core-seed-database.feature",
        "src/test/resources/features/cdi/cdi-seed-database.feature",
        "src/test/resources/features/cucumber/cucumber-seed-database.feature",
        "src/test/resources/features/general/dataset-replacements.feature",
        "src/test/resources/features/general/expected-dataset.feature"
},
        plugin = "json:target/dbunit-rules.json")
public class DBUnitRulesBdd {
}
```

As cucumber doesn't work with JUnit Rules, see this issue, you won't be able to use Cucumber runner with *DBunit Rules Core* because its based on JUnit rules.

Dependencies

Here is a set of maven dependencies needed by DBUnit rules Cucumber:

Most of the dependencies, except CDI container implementation, are bring by DBUnit Rules Cucumber module transitively.

Cucumber dependencies

```
        <dependency>
            <groupId>com.github.dbunit-rules</groupId>
            <artifactId>cucumber</artifactId>
            <version>${project.parent.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency> ①
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-junit</artifactId>
            <version>1.2.4</version>
            <scope>test</scope>
        </dependency>
        <dependency> ①
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-java</artifactId>
            <version>1.2.4</version>
            <scope>test</scope>
        </dependency>
```

① You don't need to declare because it comes with DBUnit Rules Cucumber module dependency.

DeltaSpike and CDI dependency

```
        <dependency> ①
            <groupId>org.apache.deltaspike.modules</groupId>
            <artifactId>deltaspike-test-control-module-api</artifactId>
            <version>${ds.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency> ①
            <groupId>org.apache.deltaspike.core</groupId>
            <artifactId>deltaspike-core-impl</artifactId>
            <version>${ds.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency> ①
            <groupId>org.apache.deltaspike.modules</groupId>
            <artifactId>deltaspike-test-control-module-impl</artifactId>
            <version>${ds.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency> ②
            <groupId>org.apache.deltaspike.cdictrl</groupId>
            <artifactId>deltaspike-cdictrl-owb</artifactId>
            <version>${ds.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency>  ②
            <groupId>org.apache.openwebbeans</groupId>
            <artifactId>openwebbeans-impl</artifactId>
            <version>1.6.2</version>
            <scope>test</scope>
        </dependency>
```

① Also comes with DBUit Rules Cucumber.

② You can use CDI implementation of your choice.

3.3.1. Scenario: Seed database using DBUnit rules in Cucumber tests

Given

The following feature 👍 (004ms)

```
Unresolved directive in documentation.adoc -
include::../../../examples/src/test/resources/features/contacts.feature
[]
```

And

The following dataset 👍 (002ms)

```
Unresolved directive in documentation.adoc -
include::../../../examples/src/test/resources/datasets/contacts.yml[]
```

And

The following Cucumber test 👍 (000ms)

```
Unresolved directive in documentation.adoc -
include::../../../examples/src/test/java/com/github/dbunit/rules/exampl
es/cucumber/ContactFeature.java[]
```

When

The following cucumber steps are executed 👍 (000ms)

```
Unresolved directive in documentation.adoc -
include::../../../examples/src/test/java/com/github/dbunit/rules/exampl
es/cucumber/ContactSteps.java[]
```

① As the Cucumber cdi runner enables CDI, you can use injection into your Cucumber steps.

② Here we use the DBUnit Rules CDI interceptor to seed the database before step execution.

Then

The database should be seeded with the dataset content before step execution 👍 (000ms)

3.4. Dynamic data using scritable datasets

> In order to have dynamic data in datasets
> As a developer
> I want to use scripts in DBUnit datasets.

Scritable datasets are backed by JSR 223. [2: Scripting for the Java Platform, for more information access the official docs here].

3.4.1. Scenario: Seed database with groovy script in dataset

Given

Groovy script engine is on test classpath 👍 (010ms)

```
<dependency>
    <groupId>org.codehaus.groovy</groupId>
    <artifactId>groovy-all</artifactId>
    <version>2.4.6</version>
    <scope>test</scope>
</dependency>
```

And

The following dataset 👍 (000ms)

```
tweet:
  - id: "1"
    content: "dbunit rules!"
    date: "groovy:new Date()" ①
    user_id: 1
```

① Groovy scripting is enabled by `groovy:` string.

When

The following test is executed: 👍 (000ms)

```
@Test
@DataSet(value = "datasets/yml/groovy-with-date-
replacements.yml",cleanBefore = true, disableConstraints = true,
executorId = "rules-it")
public void shouldReplaceDateUsingGroovyInDataset() {
    Tweet tweet = (Tweet) emProvider.em().createQuery("select t from
Tweet t where t.id = '1'").getSingleResult();
    assertThat(tweet).isNotNull();

assertThat(tweet.getDate().get(Calendar.DAY_OF_MONTH)).isEqualTo(now.ge
t(Calendar.DAY_OF_MONTH));

assertThat(tweet.getDate().get(Calendar.HOUR_OF_DAY)).isEqualTo(now.get
(Calendar.HOUR_OF_DAY));
}
```

Then

Dataset script should be interpreted while seeding the database 👍 (000ms)

3.4.2. Scenario: Seed database with javascript in dataset

Javascript engine comes within JDK so no additional classpath dependency is necessary.

Given

The following dataset 👍 (000ms)

```
tweet:
  - id: "1"
    content: "dbunit rules!"
    likes: "js:(5+5)*10/2" ①
    user_id: 1
```

① Javascript scripting is enabled by `js:` string.

When

The following test is executed: 👍 (000ms)

```
@Test
@DataSet(value = "datasets/yml/js-with-calc-
replacements.yml",cleanBefore = true ,disableConstraints = true,
executorId = "rules-it")
public void shouldReplaceLikesUsingJavaScriptInDataset() {
    Tweet tweet = (Tweet) emProvider.em().createQuery("select t from
Tweet t where t.id = '1'").getSingleResult();
    assertThat(tweet).isNotNull();
    assertThat(tweet.getLikes()).isEqualTo(50);
}
```

Then

Dataset script should be interpreted while seeding the database 👍 (000ms)

3.5. Database assertion using expected datasets

In order to verify database state after test execution
As a developer
I want to assert database state with datasets.

3.5.1. Scenario: Database assertion with yml dataset

Given

The following dataset 👍 (002ms)

expectedUsers.yml

```
user:
  - id: 1
    name: "expected user1"
  - id: 2
    name: "expected user2"
```

When

The following test is executed: 👍 (000ms)

```
@RunWith(JUnit4.class)
public class ExpectedDataSetIt {

    @Rule
    public EntityManagerProvider emProvider =
EntityManagerProvider.instance("rules-it");

    @Rule
    public DBUnitRule dbUnitRule =
DBUnitRule.instance(emProvider.connection());


    @Test
    @DataSet(cleanBefore = true)①
    @ExpectedDataSet(value = "yml/expectedUsers.yml",ignoreCols = "id")
    public void shouldMatchExpectedDataSet() {
        EntityManagerProvider instance =
EntityManagerProvider.newInstance("rules-it");
        User u = new User();
        u.setName("expected user1");
        User u2 = new User();
        u2.setName("expected user2");
        instance.tx().begin();
        instance.em().persist(u);
        instance.em().persist(u2);
        instance.tx().commit();
    }
```

① Clear database before to avoid conflict with other tests.

Then

Test must pass because database state is as in expected dataset. 👍 (000ms)

3.5.2. Scenario: Database assertion with regular expression in expected dataset

Given

The following dataset 👍 (000ms)

expectedUsersRegex.yml

```
user:
  - id: "regex:\\d+"
    name: regex:^expected user.* #expected user1
  - id: "regex:\\d+"
    name: regex:.*user2$ #expected user2
```

When

The following test is executed: 👍 (000ms)

```
@Test
@DataSet(cleanBefore = true)
@ExpectedDataSet(value = "yml/expectedUsersRegex.yml")
public void shouldMatchExpectedDataSetUsingRegex() {
    User u = new User();
    u.setName("expected user1");
    User u2 = new User();
    u2.setName("expected user2");
    tx().begin();
    em().persist(u);
    em().persist(u2);
    tx().commit();
}
```

Then

Test must pass because database state is as in expected dataset. 👍 (000ms)

3.5.3. Scenario: Database assertion with seeding before test execution

Given

The following dataset 👍 (000ms)

user.yml

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
```

And

The following dataset 👍 (000ms)

expectedUser.yml

```
user:
  - id: 2
    name: "@dbunit"
```

When

The following test is executed: 👍 (000ms)

```
@Test
@DataSet(value = "yml/user.yml", disableConstraints = true)
@ExpectedDataSet(value = "yml/expectedUser.yml", ignoreCols = "id")
public void shouldMatchExpectedDataSetAfterSeedingDataBase() {
    tx().begin();
    em().remove(em().find(User.class,1L));
    tx().commit();
}
```

Then

Test must pass because database state is as in expected dataset. 👍 (014ms)

3.5.4. Scenario: Failling database assertion

Given

The following dataset 👍 (000ms)

expectedUsers.yml

```
user:
  - id: 1
    name: "expected user1"
  - id: 2
    name: "expected user2"
```

When

The following test is executed: 👍 (000ms)

```
@Test
@ExpectedDataSet(value = "yml/expectedUsers.yml",ignoreCols = "id")
public void shouldNotMatchExpectedDataSet() {
    User u = new User();
    u.setName("non expected user1");
    User u2 = new User();
    u2.setName("non expected user2");
    tx().begin();
    em().persist(u);
    em().persist(u2);
    tx().commit();
}
```

Then

Test must fail with following error: 👍 (000ms)

junit.framework.ComparisonFailure: value (table=USER, row=0, col=name) expected:<[]expected user1> but was:<[non]expected user1> at org.dbunit.assertion.JUnitFailureFactory.createFailure(JUnitFailureFactory.java:39) at org.dbunit.assertion.DefaultFailureHandler.createFailure(DefaultFailureHandler.java:97) at org.dbunit.assertion.DefaultFailureHandler.handle(DefaultFailureHandler.java:223) at ...

3.5.5. Scenario: Database assertion using automatic transaction

Given

The following dataset 👍 (000ms)

expectedUsersRegex.yml

```
user:
  - id: "regex:\\d+"
    name: regex:^expected user.* #expected user1
  - id: "regex:\\d+"
    name: regex:.*user2$ #expected user2
```

When

The following test is executed: 👍 (000ms)

```
@Test
@DataSet(cleanBefore = true, transactional = true)
@ExpectedDataSet(value = "yml/expectedUsersRegex.yml")
public void shouldManageTransactionAutomatically() {
    User u = new User();
    u.setName("expected user1");
    User u2 = new User();
    u2.setName("expected user2");
    em().persist(u);
    em().persist(u2);
}
```

`Transactional` attribute will make DBUnit Rules start a transaction before test and commit the transaction **after** test execution but **before** expected dataset comparison.

Then

Test must pass because inserted users are commited to database and database state matches expected dataset. 👍 (000ms)