

# Living Documentation

Version 0.13.3-SNAPSHOT

# Table of Contents

|                                                                                       |    |
|---------------------------------------------------------------------------------------|----|
| 1. Summary.....                                                                       | 1  |
| 2. Features.....                                                                      | 2  |
| 2.1. Manage database with DBUnit Rules Core .....                                     | 2  |
| 2.1.1. Scenario: Seed database using yml dataset .....                                | 2  |
| 2.2. Manage database with DBUnit Rules CDI.....                                       | 4  |
| 2.2.1. Scenario: Seed database using yml dataset .....                                | 5  |
| 2.3. Manage database with DBUnit Rules Cucumber .....                                 | 7  |
| 2.3.1. Scenario: Seed database using DBUnit rules in Cucumber tests.....              | 9  |
| 2.4. Manage database with DBUnit Rules and JUnit 5 .....                              | 14 |
| 2.4.1. Scenario: Seed database using DBUnit rules in JUnit5 tests.....                | 14 |
| 2.5. Dynamic data using scritable datasets.....                                       | 16 |
| 2.5.1. Scenario: Seed database with groovy script in dataset .....                    | 16 |
| 2.5.2. Scenario: Seed database with javascript in dataset .....                       | 17 |
| 2.6. Database assertion using expected datasets .....                                 | 18 |
| 2.6.1. Scenario: Database assertion with yml dataset.....                             | 19 |
| 2.6.2. Scenario: Database assertion with regular expression in expected dataset ..... | 20 |
| 2.6.3. Scenario: Database assertion with seeding before test execution .....          | 21 |
| 2.6.4. Scenario: Failling database assertion .....                                    | 23 |
| 2.6.5. Scenario: Database assertion using automatic transaction .....                 | 24 |

# Chapter 1. Summary

| Scenarios                                     |        |       | Steps  |        |         |         |           |         |       | Features: 6 |        |
|-----------------------------------------------|--------|-------|--------|--------|---------|---------|-----------|---------|-------|-------------|--------|
| 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     | 000ms       | passed |
| Manage database with DBUnit Rules CDI         |        |       |        |        |         |         |           |         |       |             |        |
| 1                                             | 0      | 1     | 4      | 0      | 0       | 0       | 0         | 0       | 4     | 01s 221ms   | passed |
| Manage database with DBUnit Rules Cucumber    |        |       |        |        |         |         |           |         |       |             |        |
| 1                                             | 0      | 1     | 5      | 0      | 0       | 0       | 0         | 0       | 5     | 000ms       | passed |
| Manage database with DBUnit Rules and JUnit 5 |        |       |        |        |         |         |           |         |       |             |        |
| 1                                             | 0      | 1     | 3      | 0      | 0       | 0       | 0         | 0       | 3     | 000ms       | passed |
| Dynamic data using scribable datasets         |        |       |        |        |         |         |           |         |       |             |        |
| 2                                             | 0      | 2     | 7      | 0      | 0       | 0       | 0         | 0       | 7     | 000ms       | passed |
| Database assertion using expected datasets    |        |       |        |        |         |         |           |         |       |             |        |
| 5                                             | 0      | 5     | 16     | 0      | 0       | 0       | 0         | 0       | 16    | 001ms       | passed |
| Totals                                        |        |       |        |        |         |         |           |         |       |             |        |
| 11                                            | 0      | 11    | 39     | 0      | 0       | 0       | 0         | 0       | 39    | 01s 223ms   |        |

# Chapter 2. Features

## 2.1. 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](#).

### Dependencies

To use it just add the following maven dependency:

```
<dependency>  
  <groupId>com.github.dbunit-rules</groupId>  
  <artifactId>core</artifactId>  
  <version>0.13.3-SNAPSHOT</version>  
  <scope>test</scope>  
</dependency>
```

#### 2.1.1. Scenario: Seed database using yaml dataset

*Given*

The following junit rules 🍌 (000ms)

```
@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!"
    date: "[DAY,NOW]"
    user_id: 1
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();
    assertNotNull(user);
    assertEquals(1, user.getId());
    assertNotNull(user.getTweets().hasSize(1));
    Tweet tweet = user.getTweets().get(0);
    assertNotNull(tweet);
    Calendar date = tweet.getDate();
    Calendar now = Calendar.getInstance();

    assertEquals(date.get(Calendar.DAY_OF_MONTH), now.get(Calendar.
DAY_OF_MONTH));
}
```



Source code of the above example can be [found here](#).

*Then*

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

## 2.2. 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.

## Dependencies

To use this module just add the following maven dependency:

```
<dependency>
  <groupId>com.github.dbunit-rules</groupId>
  <artifactId>cdi</artifactId>
  <version>0.13.3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

### 2.2.1. Scenario: Seed database using yaml dataset

*Given*

DBUnit interceptor is enabled in your test beans.xml: 🍌 (01s 220ms)

*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.DBUnitInterceptorImpl</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: "@realpestando"
  - id: 2
    name: "@dbunit"
tweet:
  - id: abcdef12345
    content: "dbunit rules!"
    date: "[DAY,NOW]"
    user_id: 1
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]
```



Source code of the above example can be [found here](#).

Then

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

## 2.3. 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 cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

/**
 * Created by rmpestano on 4/17/16.
 */
@RunWith(Cucumber.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/junit5/junit5-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, but you can use DataSetExecutor in `@Before`, see [example here](#).

## 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.

```

<dependency>
  <groupId>com.github.dbunit-rules</groupId>
  <artifactId>cucumber</artifactId>
  <version>0.13.3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>

```

### *Cucumber dependencies*

```
<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*

- ① Also comes with DBUnit Rules Cucumber.
- ② You can use CDI implementation of your choice.

To use this module just add the following maven dependency:

## **2.3.1. Scenario: Seed database using DBUnit rules in Cucumber tests**

*Given*

The following feature 👍 (000ms)

Feature: Contacts test

As a user of contacts repository

I want to crud contacts

So that I can expose contacts service

Scenario Outline: search contacts

Given we have a list of contacts

When we search contacts by name "<name>"

Then we should find <result> contacts

Examples: examples1

| name     | result |
|----------|--------|
| delta    | 1      |
| sp       | 2      |
| querydsl | 1      |
| abcd     | 0      |

Scenario: delete a contact

Given we have a list of contacts

When we delete contact by id 1

Then we should not find contact 1

*And*

The following dataset 🍌 (000ms)

```
contact:
- id: 1
  name: "deltaspike"
  email: "users@deltaspike.apache.org"
  company_id: 1
- id: 2
  name: "querydsl"
  email: "info@mysema.com"
  company_id: 2
- id: 3
  name: "Spring"
  email: "spring@pivotal.io"
  company_id: 3

company:
- id: 1
  name: "Apache"
- id: 2
  name: "Mysema"
- id: 3
  name: "Pivotal"
- id: 4
  name: "Google"
```

*And*

The following Cucumber test 🍌 (000ms)

```
package com.github.dbunit.rules.examples.cucumber;

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

@RunWith(CdiCucumberTestRunner.class)
@CucumberOptions(
    features = {"src/test/resources/features/contacts.feature"},
    plugin = {"json:target/cucumber.json"}
    //glue = "com.github.dbunit.rules.examples.glues"
)
public class ContactFeature {
}
```

## When

The following cucumber steps are executed 🍌 (000ms)

```
package com.github.dbunit.rules.examples.cucumber; ①

import com.github.dbunit.rules.api.dataset.DataSet;
import com.github.dbunit.rules.cdi.api.DBUnitInterceptor;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.example.jpdomain.Contact;
import org.example.jpdomain.Contact_;
import org.example.service.deltaspike.ContactRepository;

import javax.inject.Inject;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

@DBUnitInterceptor
public class ContactSteps {

    @Inject
    ContactRepository contactRepository; ①

    Long count;
```

```

    @When("^we search contacts by name \"([^\"]*)\"$")
    public void we_search_contacts_by_name_(String name) throws
    Throwable {
        Contact contact = new Contact();
        contact.setName(name);
        count = contactRepository.countLike(contact, Contact_.name);
    }

    @Then("^we should find (\\d+) contacts$")
    public void we_should_find_result_contacts(Long result) throws
    Throwable {
        assertEquals(result, count);
    }

    @Given("^we have a list of contacts$")
    @DataSet("datasets/contacts.yml") ②
    public void given() {
        assertEquals(contactRepository.count(), new Long(3));
    }

    @When("^we delete contact by id (\\d+)$")
    public void we_delete_contact_by_id(long id) throws Throwable {
        contactRepository.remove(contactRepository.findBy(id));
    }

    @Then("^we should not find contact (\\d+)$")
    public void we_should_not_find_contacts_in_database(long id) throws
    Throwable {
        assertNull(contactRepository.findBy(id));
    }
}

```

- 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.



Source code for the example above can be [found here](#).

*Then*

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

## 2.4. Manage database with DBUnit Rules and JUnit 5

In order to manage database state in [JUnit 5](#) integration tests

As a developer

I want to use DBUnit along side my JUnit 5 tests.

DBUnit is enabled in JUnit 5 tests through an [extension](#) named **DBUnitExtension**.

### Dependencies

To use the extension just add the following maven dependency:

```
<dependency>
  <groupId>com.github.dbunit-rules</groupId>
  <artifactId>junit5</artifactId>
  <version>0.13.3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

#### 2.4.1. Scenario: Seed database using DBUnit rules in JUnit5 tests



## Given

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!"
    date: "[DAY,NOW]"
    user_id: 1
follower:
  - id: 1
    user_id: 1
    follower_id: 2
```

## When

The following junit5 test is executed 🍌 (000ms)

```
Unresolved directive in documentation.adoc -
include:../../../../junit5/src/test/java/com/github/dbunit/junit5/DBUnitJ
Unit5Test.java[tags=declaration;connectionField;test]
```

- ① Enables DBUnit;
- ② JUnit 5 runner;
- ③ As JUnit5 requires **Java8** you can use lambdas in your tests;
- ④ DBUnitExtension will get connection by reflection so just declare a field or a method with **ConnectionHolder** as return type.



Source code of the above example can be [found here](#).

## Then

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

## 2.5. 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](#)].

### 2.5.1. Scenario: Seed database with groovy script in dataset

*Given*

Groovy script engine is on test classpath 🍌 (000ms)

```
<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)

### 2.5.2. 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)

## 2.6. 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.

### 2.6.1. Scenario: Database assertion with yml dataset

*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)

```
@RunWith(JUnit4.class)
@DBUnit(cacheConnection = true)
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 = "yaml/expectedUsers.yaml", 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)

## 2.6.2. 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 = "yaml/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)

### 2.6.3. 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 = "yaml/user.yml", disableConstraints = true)
@ExpectedDataSet(value = "yaml/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. 🍌 (000ms)



## 2.6.4. Scenario: Failing 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 = "yaml/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(JUnitFailur
eFactory.java:39)                         at
org.dbunit.assertion.DefaultFailureHandler.createFailure(Default
FailureHandler.java:97)                   at
org.dbunit.assertion.DefaultFailureHandler.handle(DefaultFailure
Handler.java:223) at ...
```

### 2.6.5. 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, executorId =
"TransactionIt")
@ExpectedDataSet(value = "yml/expectedUsersRegex.yml")
@DBUnit(cacheConnection = true)
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 committed to database and database state matches expected dataset. 🍌 (000ms)