

## Concordion

Tomo Popovic, tp0x45 [at] gmail.com

Concordion is an open source tool for writing automated acceptance tests in Java development environment. The main advantages of Concordion are based on its clean concept and simplicity. It is very easy to install, learn, and use.

**Web site:** <http://www.concordion.org>

**Version Tested:** Concordion 1.4.2

**License & Pricing:** Free, under Apache License, Version 2.0

**Support:** website and users group on yahoo: <http://tech.groups.yahoo.com/group/concordion/>

### Introduction

Concordion is a very powerful tool for writing and managing automated acceptance tests in Java projects. Concordion directly integrates with JUnit, which allows for easy use with IDE of your choice (Netbeans, Eclipse, IntelliJ IDEA). One of the most appealing features is that Concordion uses acceptance tests specifications in native language, which allows use of Concordion for requirements management.

### Installation

Installation of Concordion is very simple. You basically need to download and insert Concordion JAR file (comes with three dependency libraries provided). In NetBeans all the JARs go into the section for Test Libraries as shown in Figure 1.



Figure 1. Simple installation: inserting Concordion JAR files into the project

Alternatively, if you are using Maven, Concordion can be referenced in the POM file:

```
<dependency>
  <groupId>org.concordion</groupId>
  <artifactId>concordion</artifactId>
  <version>1.4.2</version>
</dependency>
```

Figure 2. Referencing Concordion using Maven

### Using Concordion

Using Concordion assumes that developers understand and entertain the idea of active software specifications. Each feature or behavior needs to be specified, implemented, and verified by the means of active specifications and their connection to the system under development.

An active specification in Concordion consists of two key parts:

1. A nicely written requirement document describing desired functionality (XHTML). The XHTML specifications contain descriptions of the functionality illustrated with acceptance test examples. The examples data is marked using simple HTML tags.
2. Acceptance tests are written in Java and called fixture code. Tests are coded implementing a Concordion extension of a standard JUnit test case. Fixture code finds example data by marked by tags and use them to verify the system under development.

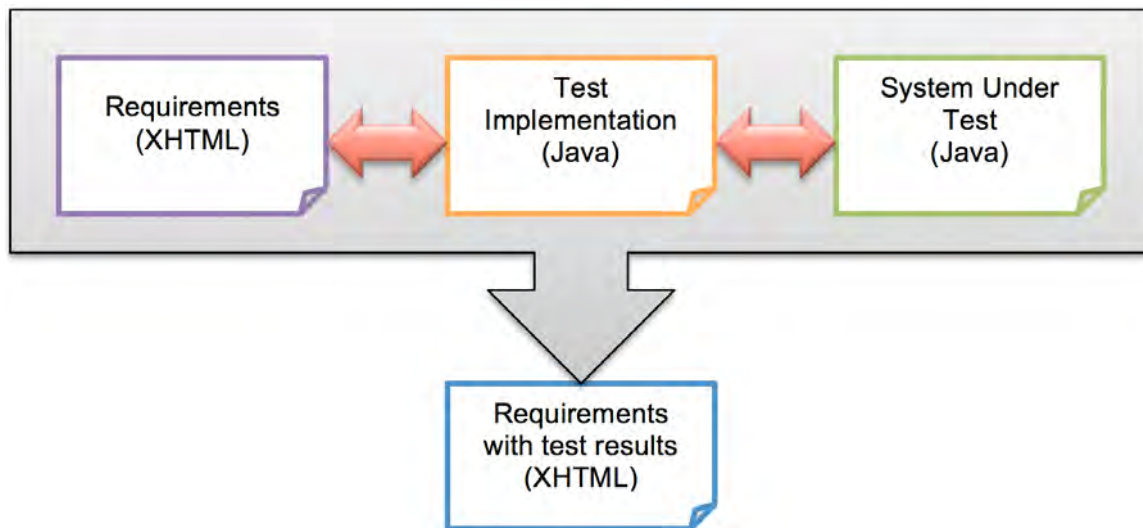


Figure 3. Concept of active specifications

The concept of using Concordion is illustrated in Figure 3. Acceptance tests are specified using native language and organized in Requirements (XHTML) files. Tests are implemented in Java and they connect examples from requirements with Java code of the system being tested. Test code is connected with the requirements through XHTML tags, which contain Concordion commands. This will be illustrated later in the article. Running tests in Concordion results in output XHTML files that combine the original specification and test results. Successful tests are highlighted “green” and unsuccessful with “red”.

This concept in Concordion is called active specifications due to a fact that test implementation links specifications and the system. Any change in the system will result in failed tests, which will remind us that we have to update the specification. Therefore, specifications never get old.

In order for Concordion to work, your project file structure needs to follow certain rules. In the example shown in Figure 4, we organize specifications, and tests implementation files into a package structure, here named **exampleapp.spec**.

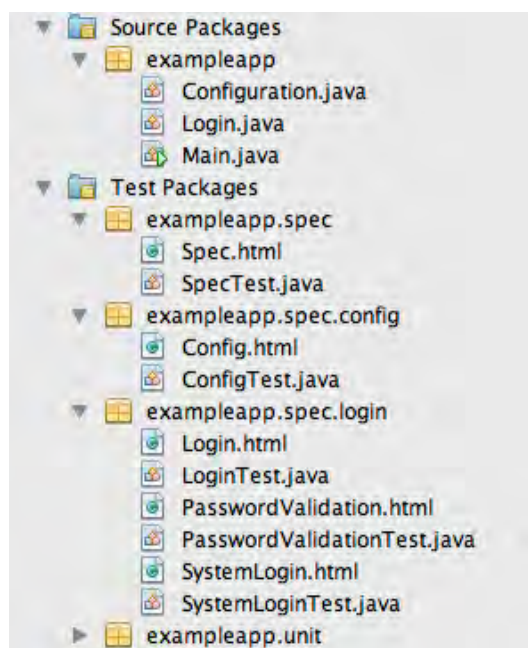


Figure 4. Organizing active specifications within Java project

As we can see in the example, other specifications and tests can be organized in sub-folders, which helps navigation. The folders structure also allows for easier navigation through the output files, which is implemented using “breadcrumbs”.

### Simple example

To write specifications in Concordion, we use fairly simple XHTML syntax. In each specification document we need to use the “concordion” namespace at the top of the XHTML file. Please refer to the top of the example in Figure 5.

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
    <title>Config</title>
  </head>
  <body>
    <h1>REQ-003 Config</h1>
    <p>
      Provides feature for adding two integers. For example
      <span concordion:set="#a">2</span> plus
      <span concordion:set="#b">5</span> should give
      <span concordion:assertEquals="getTestResult(#a,#b)">7</span>.
    </p>
  </body>
</html>
```

Figure 5. A simple example of active specifications: Config.html file

Further in the example, we use simple XHTML tags to mark specific parts of the example and connect them to our test implementation code, which is in Concordion also called fixture code. Variables **#a** and **#b** are set to 2 and 5 inside of the sentence “For example 2 plus 5 should give 7” using **concordion:set** command. Another tag is specifying **concordion:assertEquals** command, which as a parameter in this case invokes internal method implemented in the corresponding test class. For more information on Concordion tags and commands please refer to Concordion website, which provides an excellent tutorial. Java fixture code for this active

specification is given in Figure 6.

```
package exampleapp.spec.config;

import exampleapp.Configuration; // connects to the System Being Tested
import org.concordion.integration.junit3.ConcordionTestCase;

public class ConfigTest extends ConcordionTestCase {

    public int getResult(int a, int b) {
        Configuration conf = new Configuration();
        return conf.addTwoIntegers(a,b);
    }
}
```

Figure 6. Test implementation (fixture code): ConfigTest.java

The implementation class uses name same as the XHTML specification, but with suffix Test, in this case **Config.html** and **ConfigTest.java**. The test connects the actual system being tested with the XHTML active specification. An instance of Configuration class is created, and then its method that adds two numbers is called. The result is passed back to the active specification. Upon executing the tests, the Concordion generates an output XHTML file, with inserted red or green highlights around the test results.



Figure 7. Test results: number 7 highlighted “green”

### Providing tests in tables

Sometimes we need to run several test cases in order verify the desired behavior. In automated acceptance test tools this is typically done via tables. Concordion is not an exception. We can specify a test data set using standard XHTML tables and then use Concordion command set applied to table headers, in form of tags, to mark the table data for use in testing. Concordion will grab the test data from each row in the table, run the acceptance test and compare the results against the expected output. This feature is very useful when there is a need to run several testing data sets.

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
    <title>System Login</title>
  </head>
  <body>
    <h1>REQ-001 System Login</h1>
    <p>
      The system shall provide system logon function that will be used when
      a user attempts to use the system. The user needs to provide credentials
      in a form of username and password pair.
    </p>
    <div class="example">
      <h3>Examples</h3>
      <p>
        For the purpose of demonstration the system under test contains a
        system login function with username and password hardcoded to
        "johndoe" and "123abc!@#" respectively. All other combinations
        should fail.
      </p>
      <table concordion:execute="#valid=systemLogin(#username, #password)">
        <tr>
          <th concordion:set="#username">Username</th>
          <th concordion:set="#password">Password</th>
          <th concordion:assertEquals="#valid">Sucess</th>
        </tr>
        <tr>
          <td>john</td>
          <td>doe123abc!@#</td>
          <td>no</td>
        </tr>
        <tr>
          <td>admin</td>
          <td>admin</td>
          <td>no</td>
        </tr>
        <tr>
          <td>johndoe</td>
          <td>123abc!@#</td>
          <td>yes</td>
        </tr>
      </table>
    </div>
    <h2>Further Details</h2>
    <ul>
      <li>How to create user and password?</li>
      <li>Username restrictions and validation?</li>
      <li><a href="PasswordValidation.html">
        Password restrictions and validation?</a></li>
      <li>Can email be used instead of username?</li>
    </ul>
  </body>
</html>
```

Figure 8. An example of test data in a form of table – PasswordValidation.html

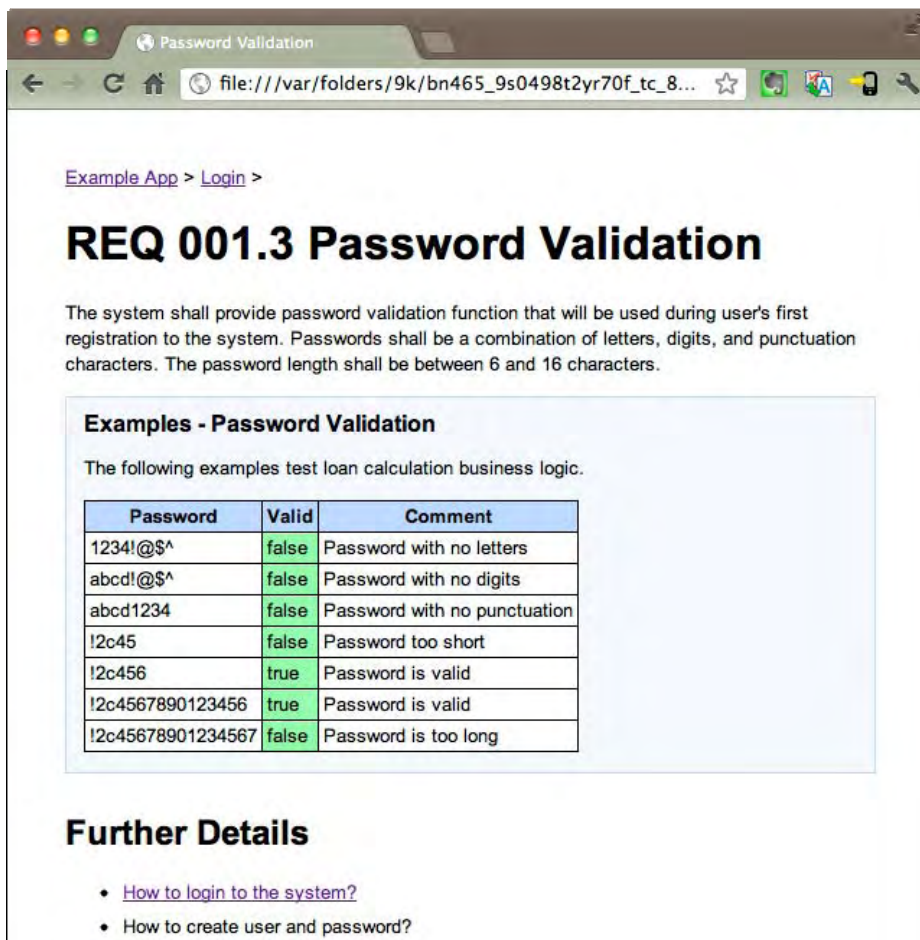
```
package exampleapp.spec.login;

import org.concordion.integration.junit3.ConcordionTestCase;
import exampleapp.Login; // system under design

public class PasswordValidationTest extends ConcordionTestCase {

    public boolean isValid(String password) {
        Login login = new Login();
        return login.validatePassword(password);
    }
}
```

Figure 9. Test implementation (fixture code) - PasswordValidationTest.java



Example App > Login >

## REQ 001.3 Password Validation

The system shall provide password validation function that will be used during user's first registration to the system. Passwords shall be a combination of letters, digits, and punctuation characters. The password length shall be between 6 and 16 characters.

### Examples - Password Validation

The following examples test loan calculation business logic.

Password	Valid	Comment
1234!@\$^	false	Password with no letters
abcd!@\$^	false	Password with no digits
abcd1234	false	Password with no punctuation
!2c45	false	Password too short
!2c456	true	Password is valid
!2c4567890123456	true	Password is valid
!2c45678901234567	false	Password is too long

### Further Details

- [How to login to the system?](#)
- How to create user and password?

Figure 10. Tabular test output: successful runs highlighted “green”

An additional benefit of using XHTML files for specifications is that we can use hyperlinks between the specifications. In the example above, this is illustrated with “Further Details” section that links to Password restrictions and validation.

## Suite of Tests

It is very wise to organize and structure our specifications and tests nicely. In Concordion, this comes very naturally for Java developers to organize specifications and tests into packages. In addition, Concordion offers a very nice way of grouping related tests into groups or subgroups and, therefore, create test suites. As shown in the example in Figure 11, a test suite can be created by simply referencing other XHTML test specifications.

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
<title>Example App</title>
  </head>
  <body>
    <h1>Example App</h1>
    <p>
      <a concordion:run ="concordion" href="login/Login.html">
        REQ-001 System Login</a>
    </p>
    <p>
      REQ-002 Browse System Events
    </p>
    <p>
      <a concordion:run ="concordion" href="config/Config.html">
        REQ-003 Update System Settings</a>
    </p>
  </body>
</html>
```

Figure 11. Suite of tests generated by referencing XHTML files (Spec.html)

```
package exampleapp.spec;

import org.concordion.integration.junit4.ConcordionRunner;
import org.junit.runner.RunWith;

@RunWith(ConcordionRunner.class)
public class SpecTest {

}
```

Figure 12. Empty class that implies running of tests in suite (SpecTest.java)

An empty class shown in Figure 12 implies running of tests in the suite. Please note that tests in the suite need to be organized in proper sub-packages as shown previously in Figure 4. This organization of files also produces nice “breadcrumb” navigation links at the top of the output pages (please refer to the top of the page in Figure 10 above). Hyperlinks in the result file will be highlighted red or green depending on the outcome of the test run (Figure 13).

## Extensions

Concordion extensions allow for adding new functionalities. Extensions enable users to add their own commands, listen to events, and modify the Concordion output. The extensions are installed from separate JAR. The installation and use of extensions is beyond the scope of this article, but it is important to mention them. Please refer to Concordion website for more information.

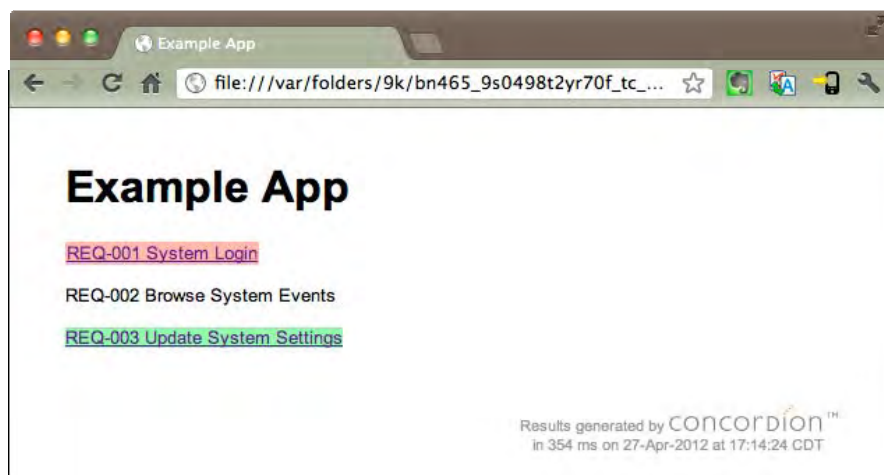


Figure 13. Test suite: highlighted hyperlinks indicate success/fail of tests

## Conclusions

Concordion is a very interesting open source tool for automating acceptance testing in your Java projects. It is very easy to install and it has fairly fast learning curve. It can easily be used with various IDEs. If used properly and pragmatically, it can be a very good tool for managing the requirements specifications as well. Since the specifications are linked with the system, any change in the system or specifications will result in failed tests, which will prompt us to keep the specifications up-to-date and in sync with the system. Therefore, our specifications never go stale.

Concordion was originally developed for Java, but now there are also versions for .NET, Python, Scala, and Ruby. Please check the Concordion website for latest development.

## Further reading

- Concordion website: <http://www.concordion.org>
- Users group: <http://tech.groups.yahoo.com/group/concordion/>
- Lisa Crispin, Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams”, Addison-Wesley, 2009