

Citrus Administration UI

Christoph Deppisch

Version 1.0.0, 2017-06-27

citrus-admin

1. Introduction	2
1.1. Features	2
1.1.1. Stable	2
1.1.2. Under construction	3
1.2. Limitations	3
1.2.1. Java 8	3
1.2.2. Maven	3
1.2.3. Version	4
1.2.4. Browsers	4
1.2.5. Java DSL	4
2. Installation	5
3. Project	6
3.1. New project	6
3.1.1. Git clone	7
3.1.2. Maven archetype	8
3.2. Open project	8
3.2.1. Customize settings	12
3.3. Dashboard	14
3.4. Settings	17
3.4.1. General	18
3.4.2. Build configuration	19
3.4.3. Modules	19
3.4.4. Admin connector	22
4. Configuration	25
4.1. Endpoint configuration	25
4.2. Spring application context	35
4.3. Spring beans	36
4.4. Global Variables	36
4.5. Schema Repositories	36
4.6. Namespace Context	36
4.7. Data Dictionaries	36
4.8. Functions	36
4.9. Validation Matcher	36
5. Tests	37
5.1. Search tests	37
5.2. List tests	38
5.3. Editor	40
5.4. Execute tests	50

5.5. Test designer	56
5.6. Test reporting	60
6. Links & Further reading.....	63



Chapter 1. Introduction

This is a web administration user interface for the integration test framework Citrus www.citrusframework.org written in Angular2. Major functionality objectives are project and configuration management as well as test execution and reporting.

1.1. Features

The Citrus administration UI is there to help you manage configuration, modules and tests in your project. The user interface is able to give you a hand when it comes to managing Citrus messaging components and other configuration items such as (endpoints, validation Matcher, functions, schema repositories and so on).

The UI is browser based and is able to open and read your Citrus project. The administration pages provide access to the test cases so you can view and execute them. Of course you will also be able to review the test results. The administration web UI is not there to eliminate your favorite IDE (IntelliJ, Eclipse or whatever)! The UI is a helping instrument for getting in touch with Citrus and its concepts and works side by side with your local Java IDE as well as other text editors of your choice.

Also the UI is helpful when executing the Citrus integration tests in different stages (test, acceptance, explorative) of your release process. There is not always a full capable development environment available for executing integration tests. You can run the Citrus administrative UI as Docker container or Kubernetes pod in order to make the tests portable to your containerized test environment.

Now lets have a quick look at the feature set.

1.1.1. Stable

Consider following features to be in a stable state:

- [Project](#)
 - [Open project](#)
 - [New project](#)
 - [Dashboard](#)
 - [Settings](#)
- [Configuration](#)
 - [Endpoints](#)
 - [Global Variables](#)
 - [Namespace Context](#)
 - [Data Dictionaries](#)
 - [Functions](#)

- [Validation Matcher](#)
- [Schema Repositories](#)
- [Tests](#)
 - [Search tests](#)
 - [List tests](#)
 - [Editor](#)
 - [Execute tests](#)

1.1.2. Under construction

Consider following features to be under construction:

- Project
 - [Statistics](#)
- Tests
 - [Test designer](#)
 - [Reporting](#)

Please let us know if you are missing a feature and/or like to vote for features.

1.2. Limitations

NOTE: This project is still still under construction!

The administration UI is stable not yet finished. Some features are still under construction. Some aspects are simply not covered yet.

Following from that we have to deal with some limitations and trade offs until the project emerges (hopefully with the help of the community, keeping our fingers crossed!). However the administration UI is usable and most features are considered to be stable.

Please consider following limitations that we have right now:

1.2.1. Java 8

The administration UI is implemented using Java 8. You need at least Java 8 runtime in order to run the Citrus admin web application on your machine.

1.2.2. Maven

At the moment we are limited to Citrus projects using Maven as build tool. Gradle projects are currently not supported. The build tool used is quite important as the administration UI is reading the build information and uses Maven to execute test cases. We are working on Gradle support, please stay tuned and maybe poke us when things are not evolving.

1.2.3. Version

Your project should use Citrus version 2.6 or higher in order to be able to work with the administration UI. Earlier Citrus versions might work too but are not tested and will not get support with bugfixes.

In addition to that we expect you to use the Citrus annotations when declaring test methods. The administration UI is looking for all tests in your project. It is required that you use **@CitrusTest** and **@CitrusXmlTest** annotations on your test methods. Otherwise the test cases will not be found and displayed.

1.2.4. Browsers

At this early state we do not support browsers other than Chrome. This does not mean that other browsers are not working with the administration UI but the features are not tested yet with other browsers. The Citrus development team is using Chrome so you can be sure that errors related to browser incompatibility will be fixed very soon for Chrome.

1.2.5. Java DSL

Citrus provides both XML and Java DSL for writing test cases. The administration UI is definitely able to read your XML test cases. The UI should also be able to read and manage your Java DSL test cases but there might be some limitations when reading your Java DSL code especially when you use object marshalling/unmarshalling as well as highly customized methods.

Please be curious and find out whether the current UI state is working with your project. And please let us know when there is something wrong.

Chapter 2. Installation

The Citrus administration UI is a web application that uses Spring boot and Angular2. First of all download the latest distribution which is a Java WAR file located at labs.consol.de/maven/repository:

```
curl -o citrus-admin.war  
https://labs.consol.de/maven/repository/com/consol/citrus/citrus-admin-  
web/1.0.0/citrus-admin-web-1.0.0-executable.war
```


Save the Java web archive to a folder on your local machine and start the Spring boot web application. The downloaded artifact should be executable from command line like this:

```
java -jar citrus-admin.war
```

You will see the application starting up. Usually you will see some console log output. The web server should start within seconds. Once the application is up and running you can open your browser and point to <http://localhost:8080>.

That's it you are ready to use the Citrus administration UI. Next thing to do is to [create](#) or [open](#) a project.

You have not selected a project yet!
Please open a project or create a new project.

 Open project

 New project

Chapter 3. Project

The basic objective of the administration UI is to manage your Citrus projects. The UI is able to open your project in order to view and edit your project configuration as well as your test cases.

3.1. New project

New projects are created in the admin UI working directory. By default this is the user home directory. You can change the working directory location by setting a system property or environment variable (`citrus.admin.working.directory` | `CITRUS_ADMIN_WORKING_DIRECTORY`).

Repository URL

master

/

Load project

Cancel

Repository URL

master

/

Load project

Cancel

3.1.1. Git clone

The administration UI is able to create new Citrus projects. The first approach would be to load the project sources from a Git repository. This repository should contain the Citrus project sources exclusively. Just give the repository URL and the UI will try to load the sources to your local file system. This is done either by git clone command (if the git binaries are available). As a fallback from that clone the UI will load the zipped project from git and unpack it on the local file system.

com.consol.citrus.mvn

citrus-quickstart

2.7.1

GroupId

ArtifactId

1.0.0-SNAPSHOT

Package

Create project

Cancel

com.consol.citrus.mvn

citrus-quickstart

2.7.1

GroupId

ArtifactId

1.0.0-SNAPSHOT

Package

Create project

Cancel

3.1.2. Maven archetype

The second approach for creating new projects is to generate the project from a Maven archetype. Citrus provides several archetypes for that. Just fill out the archetype form with the respective Maven coordinates (groupId, artifactId, version).

3.2. Open project

You can open any Citrus project. The only prerequisite is that you have access to the project home on your local machine. When the administration UI is started you have to open a project first.

ct

y

ts

cts

ct

y

ts

cts

The project home selection form is displayed automatically when no project has been selected yet. You can preselect a project home when starting the administration UI by setting a system environment variable at startup:

```
java -Dcitrus.admin.project.home=/Users/myaccount/path/tp/citrus/project/home -jar  
citrus-admin-web-1.0.0.war
```

When pre selecting a project home the project is opened automatically and the [project dashboard](#) is displayed. Now back to the project home selection if no project has been pre selected yet.

You need to specify the project home which is the root directory of your Citrus project. You can specify the complete path manually or pick the home directory over the file browser.

project home

Select Citrus project home directory and confirm.

ons

ts

ds

t

emy

ne

s

quillian-extension-spring

articles

trus

trus-admin

trus-blog

trus-demo-apps

trus-docker-images

trus-samples

logs

sample-annotation-config

sample-bakery

sample-binary


sample-bookstore

logs

src

target

test-output

 **citrus-project**

...

Select

Close

project

Close

Table 1. Project files

Path	Description
<code>\${project-home}/pom.xml</code>	Reads information from Maven POM

Path	Description
<code>\${project-home}/src/test/resources</code>	Reads XML test cases
<code>\${project-home}/src/test/resources/citrus-context.xml</code>	Reads Spring bean configuration
<code>\${project-home}/src/test/java</code>	Reads test cases

3.2.1. Customize settings

It is possible that your project uses a different folder layout for test resources and test sources (e.g. **`src/it/resources`** and **`src/it/java`**). Then the project open operation will fail with errors. We can fix this by customizing the project settings manually in prior to opening the project.

There are two different approaches to customizing the project settings: First of all you can use system properties when starting the administration UI application:

```
java -Dcitrus.admin.project.home=/Users/myaccount/path/tp/citrus/project/home
-Dcitrus.admin.java.source.directory=src/it/java
-Dcitrus.admin.xml.source.directory=src/it/resources -jar citrus-admin-web-1.0.0.war
```

You can set the following system properties:

Table 2. System properties

Property	Description
<code>server.port</code>	Web server port
<code>citrus.admin.project.home</code>	Preselect project on startup
<code>citrus.admin.root.directory</code>	System root as base of all projects (default: user home directory)
<code>citrus.admin.working.directory</code>	Base directory for new projects (default: root directory)
<code>citrus.admin.project.repository</code>	Git project repository to load on startup (default: not set)
<code>citrus.admin.java.source.directory</code>	Java sources directory (default: <code>src/test/java</code>)
<code>citrus.admin.xml.source.directory</code>	XML test sources directory (default: <code>src/test/resources</code>)
<code>citrus.admin.spring.application.context</code>	Path to Spring application context file (default: <code>src/test/resources/citrus-context.xml</code>)
<code>citrus.admin.spring.java.config</code>	Java class holding Spring bean configurations (default: <code>com.consol.citrus.CitrusEndpointConfig</code>)
<code>citrus.admin.test.base.package</code>	Base package where to add new tests (default: <code>com.consol.citrus</code>)

Property	Description
maven.home.directory	Path to Maven home that should be used in admin UI (when not set environment variable MAVEN_HOME or M2_HOME is used as default)

You can also use Spring boot properties, e.g. a custom server port:

```
java -Dserver.port=8181 -jar citrus-admin-web-1.0.0.war
```

The exact same properties are also available when set as environment variables:

Table 3. Environment variables

Environment variable	Description
CITRUS_ADMIN_PROJECT_HOME	Preselect project on startup
CITRUS_ADMIN_ROOT_DIRECTORY	System root as base of all projects (default: user home directory)
CITRUS_ADMIN_WORKING_DIRECTORY	Base directory for new projects (default: root directory)
CITRUS_ADMIN_PROJECT_REPOSITORY	Git project repository to load on startup (default: not set)
CITRUS_ADMIN_JAVA_SOURCE_DIRECTORY	Java sources directory (default: src/test/java)
CITRUS_ADMIN_XML_SOURCE_DIRECTORY	XML test sources directory (default: src/test/resources)
CITRUS_ADMIN_SPRING_APPLICATION_CONTEXT	Path to Spring application context file (default: src/test/resources/citrus-context.xml)
CITRUS_ADMIN_SPRING_JAVA_CONFIG	Java class holding Spring bean configurations (default: com.consol.citrus.CitrusEndpointConfig)
CITRUS_ADMIN_TEST_BASE_PACKAGE	Base package where to add new tests (default: com.consol.citrus)

A second approach would be to create a project settings file in your Citrus project root directory. The project settings are stored in a file called **citrus-project.json**. When you open a Citrus project for the first time the administration UI creates this project settings file automatically. But now we want to create this file manually in order to set custom directories and settings prior to opening the project. The setting file uses JSON data format and looks like this:


```

{
  "name" : "citrus-sample-project",
  "description" : "",
  "version" : "1.0.0",
  "settings" : {
    "basePackage" : "com.consol.citrus.samples",
    "citrusVersion" : "2.7.1",
    "springApplicationContext" : "src/test/resources/citrus-context.xml",
    "javaSrcDirectory" : "src/test/java/",
    "xmlSrcDirectory" : "src/test/resources/",
    "javaFilePattern" : "/*/*Test.java,/*/*IT.java",
    "xmlFilePattern" : "/*/*Test.xml,/*/*IT.xml",
    "useConnector" : true,
    "connectorActive" : true,
    "tabSize" : 2,
    "build" : {
      "@class" : "com.consol.citrus.admin.model.build.maven.MavenBuildConfiguration",
      "type" : "maven",
      "properties" : [ ],
      "testPlugin" : "maven-failsafe",
      "command" : null,
      "profiles" : "",
      "clean" : false,
      "compile" : true
    }
  }
}

```

So you can force the administration UI to use these settings when opening the project. Just create the **citrus-project.json** file in the Citrus project home directory before opening the project.

3.3. Dashboard

The dashboard gives you a quick overview of what your project looks like. Citrus reads information about your project such as name, package, description, test count, latest reports and so on.

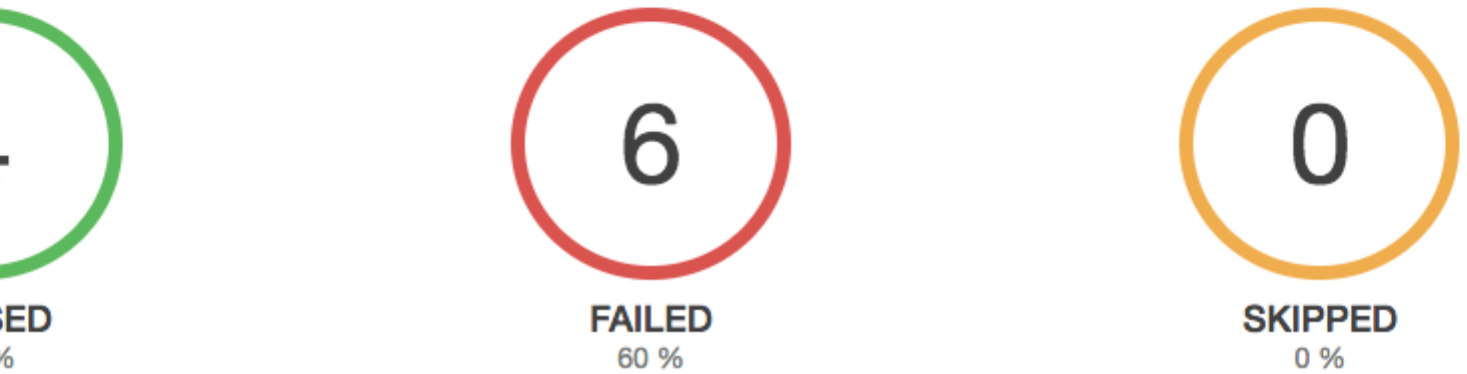
citrus-sample-incident

-sample-incident

s/christoph/Projekte/Citrus/citrus-samples/sample-incident

onsol.citrus.samples

ts 06/26/2017



citrus-sample-incident

-sample-incident

s/christoph/Projekte/Citrus/citrus-samples/sample-incident

onsol.citrus.samples

ts 06/26/2017



citrus-sample-incident

-sample-incident

s/christoph/Projekte/Citrus/citrus-samples/sample-incident

onsole.citrus.samples

ts 06/26/2017



The project dashboard is a good starting point to discover your project with all [projects settings](#) and [tests](#).

3.4. Settings

Each Citrus project has properties and settings that influence the administration UI. These properties are project names, descriptions, versions and source folders. You can review and change these project related settings with an HTML form on the project settings page.

Some project settings are read only at the moment, e.g. we do not support renaming of projects yet. If you want to rename a project or change the project version you need to do this manually in the Maven POM configuration.

If you save the project settings the administration UI will save the changes to the project settings file **citrus-project.json** which is located in your project home directory. This file uses the JSON syntax and looks like follows:

```
{
  "projectHome" : "~/Projects/Citrus/citrus-sample",
  "name" : "citrus-sample-project",
  "description" : "",
  "version" : "1.0.0",
  "settings" : {
    "basePackage" : "com.consol.citrus",
    "citrusVersion" : "2.6",
    "springApplicationContext" : "src/it/resources/citrus-context.xml",
    "javaSrcDirectory" : "src/test/java/",
    "xmlSrcDirectory" : "src/test/resources/",
    "javaFilePattern" : "**/*Test.java,**/*IT.java",
    "xmlFilePattern" : "**/*Test.xml,**/*IT.xml",
    "useConnector" : true,
    "build" : {
      "@class" : "com.consol.citrus.admin.model.build.maven.MavenBuildConfiguration",
      "type" : "maven",
      "properties" : [ ],
      "testPlugin" : "maven-failsafe",
      "clean" : false,
      "compile" : true,
      "command" : null,
      "profiles" : null
    }
  }
}
```

3.4.1. General

Each Citrus project works with Java classes and resources. These files are located in project folders inside the Maven project. Citrus admin is working with these defaults:

- src/test/java/ folder for Java test classes
- src/test/resources/ folder for test resources (e.g. configuration files)
- /**/*Test.java,**/*IT.java file pattern for Java test classes
- /**/*Test.xml,**/*IT.xml file pattern for XML test cases

You can customize these settings according to your project setup.

3.4.2. Build configuration

The administration web UI is able to execute tests. This test execution is done by calling the Maven build lifecycle for the opened project. You can adjust the build settings accordingly. By default Citrus admin uses the **maven-failsafe** to execute the Citrus tests. This causes Citrus to call

```
mvn compile integration-test
```

This executes all Citrus test cases. You can change this to **maven-surefire** so the Maven command looks like this:

```
mvn compile test
```

In case you need to activate Maven profiles during the build you need to add those profiles to the build settings. Each profile name that you save to the build settings will result in some command line argument for the Maven build like this:

```
mvn compile integration-test -PmyProfile
```

Also when some system properties should be set during the Maven build you can add those properties to the build settings, too. This results in command line arguments for the Maven command:

```
mvn compile integration-test -DmyProperty=value
```

This is how to customize the Maven build that executes the Citrus tests in a project.

NOTE: As you can see we are not speaking about Gradle build configuration here. This is simply because it is not possible to manage Gradle projects at the moment! Stay tuned for future releases to come

3.4.3. Modules

Citrus as a framework is modular. You can add and remove Citrus capabilities by adding and removing module dependencies in your project. Usually these Citrus modules are managed as Maven dependencies in your project. The Citrus administration UI is able to manage these dependencies for you.

y according to your project's needs. Citrus provides separate modules for message transports and exchanged data. By selecting to your project.

☒ **citrus-admin-connector**

name: admin-connector

version: 2.7.1

☒ **citrus-java-dsl**

name: java-dsl

version: 2.7.1

☒ **citrus-jms**

name: jms

version: 2.7.1

☒ **citrus-ws**

name: ws

version: 2.7.1

y according to your project's needs. Citrus provides separate modules for message transports and exchanged data. By selecting to your project.

☒ **citrus-admin-connector**

name: admin-connector

version: 2.7.1

☒ **citrus-java-dsl**

name: java-dsl

version: 2.7.1

☒ **citrus-jms**

name: jms

version: 2.7.1

☒ **citrus-ws**

name: ws

version: 2.7.1

eds. Citrus provides separate modules for message transports and exchanged data. By selecting the modules below Citrus will

connector	<input checked="" type="checkbox"/> citrus-java-dsl name: java-dsl version: 2.7.1	<input checked="" type="checkbox"/> citrus-jms name: jms version: 2.7.1

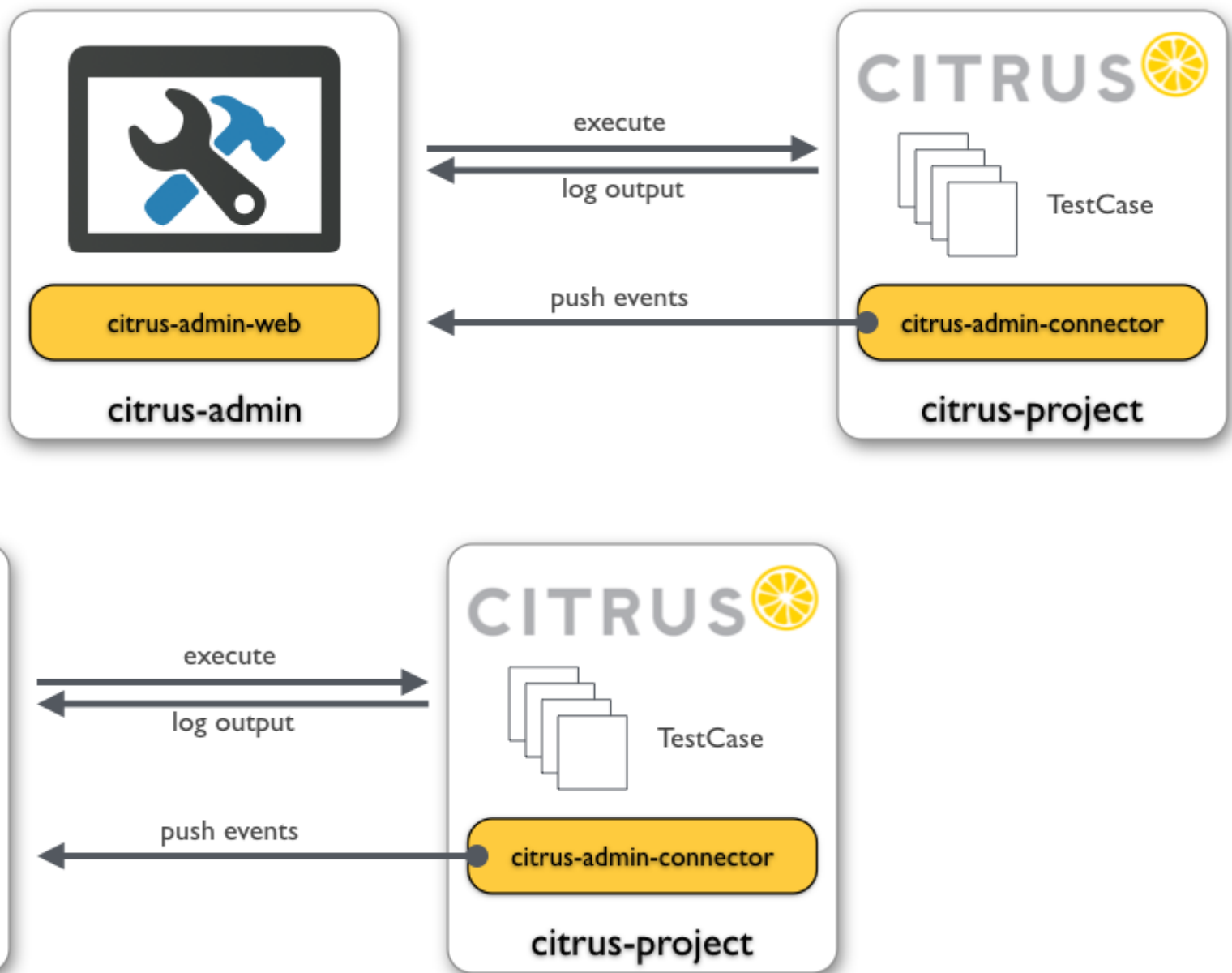
On the modules settings page you see all currently activated modules. And you get a list of available modules that you can add. Just check or uncheck the modules and the administration UI will automatically add/remove Maven dependencies in your project.

NOTE: This mechanism does not work with Gradle projects. Yet this is a feature to come soon hopefully!

3.4.4. Admin connector

In the previous chapter we have seen how to customize the project and build settings for the active project. Now when the administration UI executes some Citrus tests we can make use of a special connector library that provides detailed information about the test run and its outcome.

Basically this little helper library provides detailed information during the test run by pushing events to the admin UI.



The connector library is available from Maven central and is simply added as dependency to the target project. You can use the admin UI settings page for automatically adding this little helper to the target project. The automated connector setting will place the new Maven dependency to the project POM and add special test listeners to the Spring application context in your Citrus project.

Here is the connector Maven dependency that is added to the target project:

```
<dependency>
  <groupId>com.consol.citrus</groupId>
  <artifactId>citrus-admin-connector</artifactId>
  <version>${citrus.admin.version}</version>
</dependency>
```

Once this library is present for your project you can configure the special connector test listeners as Spring beans:

```
<bean class="com.consol.citrus.admin.connector.WebSocketPushEventListener">
  <property name="host" value="localhost"/>
  <property name="port" value="8080"/>
</bean>
```

As you can see the connector is pushing message data to the administration UI using a WebSocket API on the administration UI server. The *host* and *port* properties are customizable, default values are *localhost* and *8080*. When a test is executed the message listener will automatically connect and push messages exchanged to the administration UI.

Of course the administration UI server has to be accessible during the test run. The message listener will automatically test the server connectivity at the beginning of the test run. In case the administration UI is not accessible the message push feature is simply disabled. So you can continue to work with your Citrus project even if the administration UI is not started.

The connector will provide lots of valuable information about the running tests when activated. This is how the administration UI is able to track messages exchanged during a test run for instance. Stay tuned for more features related to the test execution and message exchange.

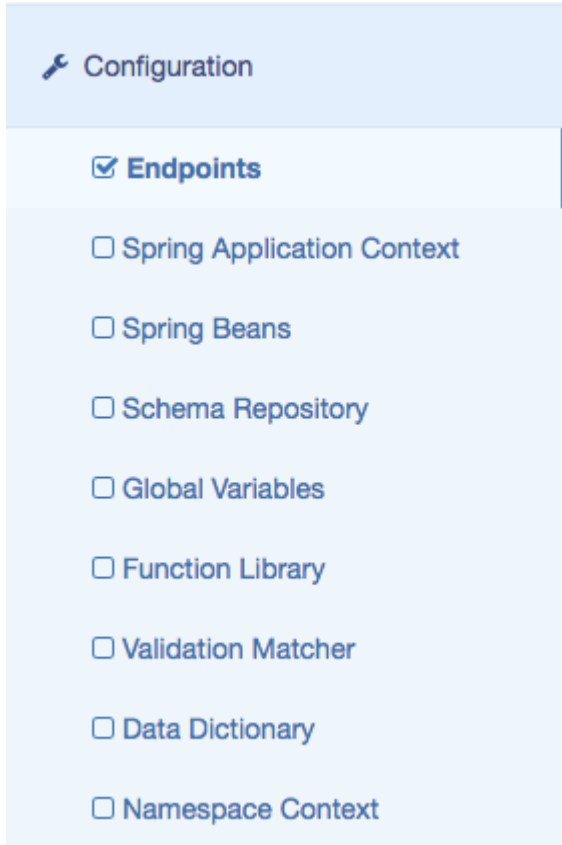
This way the admin UI is able to display runtime information of the tests such as exchanged messages, test results and so on.

As mentioned before you can automatically activate/deactivate the connector library in the project settings. Just explore the setting page for details. This will automatically add or remove the citrus-admin-connector Maven dependency for you.

NOTE: This mechanism does not work with Gradle projects. Yet this is a feature to come soon hopefully!

Chapter 4. Configuration

The Citrus components such as endpoints, variables, functions, schemas and dictionaries are configured in a Spring application context. The administration UI is able to read and change the Citrus components configuration. Each component category is represented with a separate page in the configuration section.



4.1. Endpoint configuration

Endpoints are essential in a Citrus project. They define client and server components as well as producer and consumer for different message transports.

s for sending and receiving messages to/from various endpoint destinations. Endpoint components can be of synchronous and asynchronous messages.

	Properties
Client	destination-name: JMS.Citrus.v1.FieldForceOrder pub-sub-domain: false timeout: 5000
Endpoint	destination-name: JMS.Citrus.v1.FieldForceNotification pub-sub-domain: false timeout: 5000
	request-url: http://localhost:18001/incident/IncidentManager/v1 interceptors: clientInterceptors fault-strategy: throwsException timeout: 5000
Server	port: 18002 auto-start: true root-parent-context: true interceptors: serverInterceptors timeout: 10000
	port: 18005 auto-start: true root-parent-context: true handle-mime-headers: true timeout: 10000

s for sending and receiving messages to/from various endpoint destinations. Endpoint components can be of synchronous and asynchronous messages.





















	Properties
Client	destination-name: JMS.Citrus.v1.FieldForceOrder pub-sub-domain: false timeout: 5000
Endpoint	destination-name: JMS.Citrus.v1.FieldForceNotification pub-sub-domain: false timeout: 5000
	request-url: http://localhost:18001/incident/IncidentManager/v1 interceptors: clientInterceptors fault-strategy: throwsException timeout: 5000
Server	port: 18002 auto-start: true root-parent-context: true interceptors: serverInterceptors timeout: 10000
	port: 18005 auto-start: true root-parent-context: true handle-mime-headers: true timeout: 10000

ng and receiving messages to/from various endpoint destinations. Endpoint components can be of synchronous and asynchronous

	Properties
	destination-name: JMS.Citrus.v1.FieldForceOrder pub-sub-domain: false timeout: 5000
	destination-name: JMS.Citrus.v1.FieldForceNotification pub-sub-domain: false timeout: 5000
	request-url: http://localhost:18001/incident/IncidentManager/v1 interceptors: clientInterceptors fault-strategy: throwsException timeout: 5000
	port: 18002 auto-start: true root-parent-context: true interceptors: serverInterceptors timeout: 10000
	port: 18005 auto-start: true root-parent-context: true handle-mime-headers: true timeout: 10000

First of all the list of all available endpoints in the project is displayed. Each endpoint represents a message transport such as SOAP, JMS, REST, Mail, FTP and so on. You can add new endpoints using the *New* context menu on the right. You need to chose the endpoint type first. Then a HTML form is displayed holding the endpoint settings.

New ▾

-  camel
-  channel
-  ftp-client
-  ftp-server
-  http-client
-  http-server
-  jms
-  jmx-client
-  jmx-server
-  mail-client
-  mail-server
-  rmi-client
-  rmi-server
-  ssh-client
-  ssh-server
-  vertx
-  ws-client
-  ws-server
-  websocket-client
-  websocket-server

http-client

ST

propagateError

)

00

ve

Close

http-client

ST

propagateError

)

00

ve

Close

	Name is required
	RequestId is required
	Q
	Q
	Q
	Q
	Q

Click save to add the new endpoint. Citrus is working with Spring XML configuration files. This means that the new endpoint component is saved as XML Spring bean to the basic Spring application context file. Usually this is a file located in `src/test/resources/citrus-context.xml` in your project. After you have saved the new component you will see that a new entry has been added to this file.

You can also edit endpoint components in the administration UI. Just click an existing endpoint

[illegible]

client

entHttpClient

/localhost:18001/incident/IncidentManager/v1

Interceptors

sException

Close

Manager/v1

If you save the changes Citrus will again change the Spring bean component in the XML configuration file. You can manually review the changes made. All manual changes in the Spring application context will also affect the administration UI. Just hit the reload button in your browser to reload the configuration.

4.2. Spring application context

TODO

4.3. Spring beans

TODO

4.4. Global Variables

TODO

4.5. Schema Repositories

TODO

4.6. Namespace Context

TODO

4.7. Data Dictionaries

TODO

4.8. Functions

TODO

4.9. Validation Matcher

TODO

Chapter 5. Tests

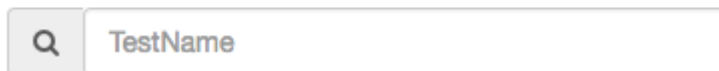
Managing all your tests is a basic thing to do with the administration UI. You are able to search for tests, open tests and execute tests with the UI.

You can do the following things regarding test management

- [Search tests](#)
- [List tests](#)
- [Open tests](#)
- [Execute tests](#)
- [Test designer](#)
- [Reporting](#)

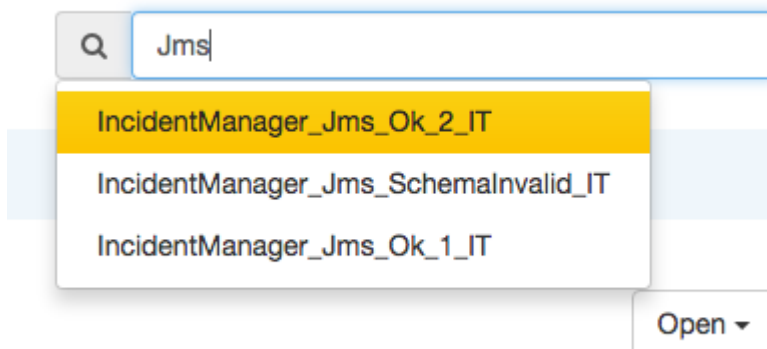
5.1. Search tests

Searching for test cases is very simple. Open the tests page and you will see a search form input field at the top right.



Citrus administration UI will search for tests in your project. These includes XML and Java DSL test cases. The search is a full text search on all test names available. So the result is always a matching list of tests.

You can open the test by selecting the test from the result list.



The test is opened and displayed in [test detail](#) panel.

5.2. List tests

Your project usually contains a lot of tests. All found tests are listed by their class and method. Also you can filter the displayed tests by the Java package. Each test can be opened in a editor for exploring further test details.

Test cases

	Class	Method
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_3
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_4
id_IT	IncidentManager_Http_IT	testIncidentManager_Http_Schema
ror_1_IT	IncidentManager_Http_IT	testIncidentManager_Http_FieldFor
ror_2_IT	IncidentManager_Http_IT	testIncidentManager_Http_FieldFor
	IncidentManager_Http_Ok_1_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Http_Ok_2_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_2
id_IT	IncidentManager_Jms_IT	testIncidentManager_Jms_Schema
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_1

	Class	Method
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_3
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_4
id_IT	IncidentManager_Http_IT	testIncidentManager_Http_Schema
ror_1_IT	IncidentManager_Http_IT	testIncidentManager_Http_FieldFo
ror_2_IT	IncidentManager_Http_IT	testIncidentManager_Http_FieldFo
	IncidentManager_Http_Ok_1_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Http_Ok_2_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_2
id_IT	IncidentManager_Jms_IT	testIncidentManager_Jms_Schema
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_1

	Class	Method
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_3
	IncidentManager_Http_IT	testIncidentManager_Http_Ok_4
	IncidentManager_Http_IT	testIncidentManager_Http_SchemaInvalid
	IncidentManager_Http_IT	testIncidentManager_Http_FieldForceError_1
	IncidentManager_Http_IT	testIncidentManager_Http_FieldForceError_2
	IncidentManager_Http_Ok_1_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Http_Ok_2_IT	testIncidentManager_Http_Ok_1
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_2
	IncidentManager_Jms_IT	testIncidentManager_Jms_SchemaInvalid
	IncidentManager_Jms_IT	testIncidentManager_Jms_Ok_1

5.3. Editor

The administration UI is able to open your test cases. Both XML and Java DSL test cases are supported.

NOTE: Java DSL test cases might cause some problems when loading the test design view. this is because we do have to make the Java DSL code interpretation more stable.

You can open the tests using the *Open* context menu. All available test cases are grouped by their package.



t cases in this project. Select a test and execute it with a run configuration.

IncidentManager_Http_Ok_3_IT

IncidentManager_Http_Ok_4_IT

IncidentManager_Http_SchemaInvalid_IT

IncidentManager_Http_FieldForceError_1_IT

IncidentManager_Http_FieldForceError_2_IT

IncidentManager_Http_Ok_1_IT

IncidentManager_Http_Ok_2_IT

IncidentManager_Jms_Ok_2_IT

IncidentManager_Jms_SchemaInvalid_IT

IncidentManager_Jms_Ok_1_IT

com.consol.o



Test cases in this project. Select a test and execute it with a run configuration.

- IncidentManager_Http_Ok_3_IT
- IncidentManager_Http_Ok_4_IT
- IncidentManager_Http_SchemaInvalid_IT
- IncidentManager_Http_FieldForceError_1_IT
- IncidentManager_Http_FieldForceError_2_IT
- IncidentManager_Http_Ok_1_IT
- IncidentManager_Http_Ok_2_IT
- IncidentManager_Jms_Ok_2_IT
- IncidentManager_Jms_SchemaInvalid_IT
- IncidentManager_Jms_Ok_1_IT**

com.consol.c

Choose a test and open it in order to see the basic test case information such as name, author, status and description.

IT

FINAL Package: **com.consol.citrus.samples.incident** Last Result: **SUCCESS**

IT

FINAL Package: **com.consol.citrus.samples.incident** Last Result: **SUCCESS**

Open



Run

incident Last Result: **SUCCESS**

Next to to the test case info you can view the test source code.

or authors.

version 2.0 (the "License");
compliance with the License.
t

ENSE-2.0

agreed to in writing, software
distributed on an "AS IS" BASIS,
OF KIND, either express or implied.
page governing permissions and

;

usXmlTest;
estNGCitrusTest;

extends AbstractTestNGCitrusTest {

HttpOk_1_IT")
Ok_1() {

or authors.

version 2.0 (the "License");
compliance with the License.
t

ENSE-2.0

agreed to in writing, software
distributed on an "AS IS" BASIS,
OF KIND, either express or implied.
page governing permissions and

;

usXmlTest;
estNGCitrusTest;

extends AbstractTestNGCitrusTest {

HttpOk_1_IT")
Ok_1() {

or authors.

version 2.0 (the "License");
compliance with the License.
t

ENSE-2.0

agreed to in writing, software
distributed on an "AS IS" BASIS,
of KIND, either express or implied.
page governing permissions and

;

usXmlTest;
estNGCitrusTest;

extends AbstractTestNGCitrusTest {

HttpOk_1_IT")

Ok_1() {

If your test uses the XML DSL to describe the test actions you can also view the XML sources.

```

network.org/schema/testcase"
framework.org/schema/jms/testcase"
framework.org/schema/ws/testcase"
framework.org/schema/samples/IncidentManager/v1"
framework.org/schema/samples/NetworkService/v1"
springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
ark.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
ark.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
ark.org/schema/testcase http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">

1_IT">

te>

</last-updated-by>
</last-updated-on>

lication via Http message transport using SOAP request message. Opens a new incident and verifies
ce as well as final incident response.</description>

citrus:randomUUID()"/>
citrus:randomNumber(6)"/>

c request message to IncidentManager via Http SOAP interface</message>

fork="true">

p://www.citrusframework.org/schema/samples/IncidentManager/v1">

'im:ticketId>
tDate('yyyy-MM-dd'T'00:00:00')</im:captured>

i:component>
went wrong with the software!</im:description>

d>
m:firstname>
lastname>
r. 38, 80995 München</im:address>

```

```

network.org/schema/testcase"
framework.org/schema/jms/testcase"
framework.org/schema/ws/testcase"
framework.org/schema/samples/IncidentManager/v1"
framework.org/schema/samples/NetworkService/v1"
springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
ark.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
ark.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
ark.org/schema/testcase http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">

1_IT">

te>

</last-updated-by>
</last-updated-on>

lication via Http message transport using SOAP request message. Opens a new incident and verifies
ce as well as final incident response.</description>

citrus:randomUUID()"/>
citrus:randomNumber(6)"/>

c request message to IncidentManager via Http SOAP interface</message>

fork="true">

p://www.citrusframework.org/schema/samples/IncidentManager/v1">

'im:ticketId>
tDate('yyyy-MM-dd'T'00:00:00')</im:captured>

i:component>
went wrong with the software!</im:description>

d>
m:firstname>
lastname>
r. 38, 80995 München</im:address>

```

```

network.org/schema/testcase"
framework.org/schema/jms/testcase"
framework.org/schema/ws/testcase"
framework.org/schema/samples/IncidentManager/v1"
framework.org/schema/samples/NetworkService/v1"
springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
ark.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
ark.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
ark.org/schema/testcase http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">

1_IT">

te>

</last-updated-by>
</last-updated-on>

lication via Http message transport using SOAP request message. Opens a new incident and verifies
ce as well as final incident response.</description>

citrus:randomUUID()"/>
citrus:randomNumber(6)"/>

t request message to IncidentManager via Http SOAP interface</message>

fork="true">

p://www.citrusframework.org/schema/samples/IncidentManager/v1">

'im:ticketId>
tDate('yyyy-MM-dd'T'00:00:00')</im:captured>

i:component>
went wrong with the software!</im:description>

d>
m:firstname>
lastname>
r. 38, 80995 München</im:address>

```

At the moment these information is read only. Stay tuned for some code editing features that might come in the future. Another representation of the test sources is the [test designer](#) view. The design view brings the test actions to a graphical representation.

Now a very powerful feature is to execute the test using the administration UI. Let's go and read about [test execution](#).

5.4. Execute tests

Test execution is a very powerful feature as it enables you to execute your tests within a browser environment almost everywhere. Just hit the *Run* button and Citrus will start a new background process that executes the test case immediately. At the moment we do only support Maven test execution. This means that a new Maven process is launched in background executing the test.

AL Package: **com.consol.citrus.samples.incident** Last Result: **SUCCESS**

100%

Http_Ok_1_IT: SUCCESS

2.7.1

(default-clean) @ citrus-sample-incident ---
te/Citrus/citrus-samples/sample-incident/target

(default) @ citrus-sample-incident ---

2java (apache-cxf-generate) @ citrus-sample-incident ---

resources (default-resources) @ citrus-sample-incident ---
iltered resources.

compile (default-compile) @ citrus-sample-incident ---
ne module!
ers/christoph/Projekte/Citrus/citrus-samples/sample-incident/target/classes

AL Package: `com.consol.citrus.samples.incident` Last Result: **SUCCESS**

100%

Http_Ok_1_IT: SUCCESS

2.7.1

```
(default-clean) @ citrus-sample-incident ---
te/Citrus/citrus-samples/sample-incident/target
```

```
(default) @ citrus-sample-incident ---
```

```
2java (apache-cxf-generate) @ citrus-sample-incident ---
```

```
resources (default-resources) @ citrus-sample-incident ---
iltered resources.
```

```
compile (default-compile) @ citrus-sample-incident ---
ne module!
ers/christoph/Projekte/Citrus/citrus-samples/sample-incident/target/classes
```

ples.incident

Last Result:

SUCCESS

100%

SUCCESS

-incident ---

incident/target

t ---

itrus-sample-incident ---

itrus-sample-incident ---

us-sample-incident ---

itrus-samples/sample-incident/target/classes

As you can see the test log output is forwarded to your browser. Also the test progress and result (success or failure) is tracked by the administration UI. In the messages table you are able to review all messages (inbound/outbound) that were part of the test run.


```
<?xml encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <http://www.citrusframework.org/schema/samples/IncidentManager/v1">
      <ticketId>a9f13b9b-81ad-4308-975a-9b6a57fadb10</im:ticketId>
      <captured>2017-06-26T00:00:00</im:captured>
      <state>NEW</im:state>
      <component>SOFTWARE</im:component>
      <description>Something went wrong with the software!</im:description>
    </http://www.citrusframework.org/schema/samples/IncidentManager/v1">
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <im:incident id="128">
      <im:firstname>Christoph</im:firstname>
      <im:lastname>Deppisch</im:lastname>
      <im:address>Franziskanerstr. 38, 80995 München</im:address>
    </im:incident>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ng="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

"http://www.citrusframework.org/schema/samples/IncidentManager/v1">

>

Id>a9f13b9b-81ad-4308-975a-9b6a57fadb10</im:ticketId>

ed>2017-06-26T00:00:00</im:captured>

NEW</im:state>

ent>SOFTWARE</im:component>

ption>Something went wrong with the software!</im:description>

t>

>

128</im:id>

ame>Christoph</im:firstname>

me>Deppisch</im:lastname>

s>Franziskanerstr. 38, 80995 München</im:address>

r>

ent>

```
<?xml version='1.0' encoding='UTF-8'>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<soap:Body>
  <im:IncidentManager xmlns:im="http://schemas.xmlsoap.org/soap/envelope/">
```

```
    <im:ticketId>db10</im:ticketId>
```

```
</im:IncidentManager>
</soap:Body>
</soap:Envelope>
```

```
<im:description>Software!</im:description>
```

```
<im:address>en</im:address>
```

The message panel displays all inbound and outbound messages. Click on the message row to see the message content details. The mechanism for tracking inbound and outbound messages during a test run is done with either the [admin connector library](#) that you need to activate in your project. If for some reason you are not able to activate the connector library in your project the administration UI will try to read the messages from the normal Citrus logging output. This of course is only working if you have logging enabled in addition to using at least with logging level *DEBUG*.

If none of these approaches is working for you the admin UI will not display exchanged messages after the test run.

5.5. Test designer

The next feature is quite experimental. In the test design view Citrus tries to give you a graphical representation of the test actions in your test. Each test action is displayed as a graphical node. If you enter the action with the mouse or if you click on a test action node some more details are displayed.







NOTE: The test design view is not complete for all test actions. Some actions may not be displayed or may have limited display.

5.6. Test reporting

The administration UI is able to read and parse basic TestNG and JUnit reports. Usually these reports are written after each test run to the build output directory of your project. The admin UI will automatically find those reports and display the results to you.

st test results

r the active



PASSED
40 %



FAILED
60 %



SKIPPED
0 %

st test results

r the active



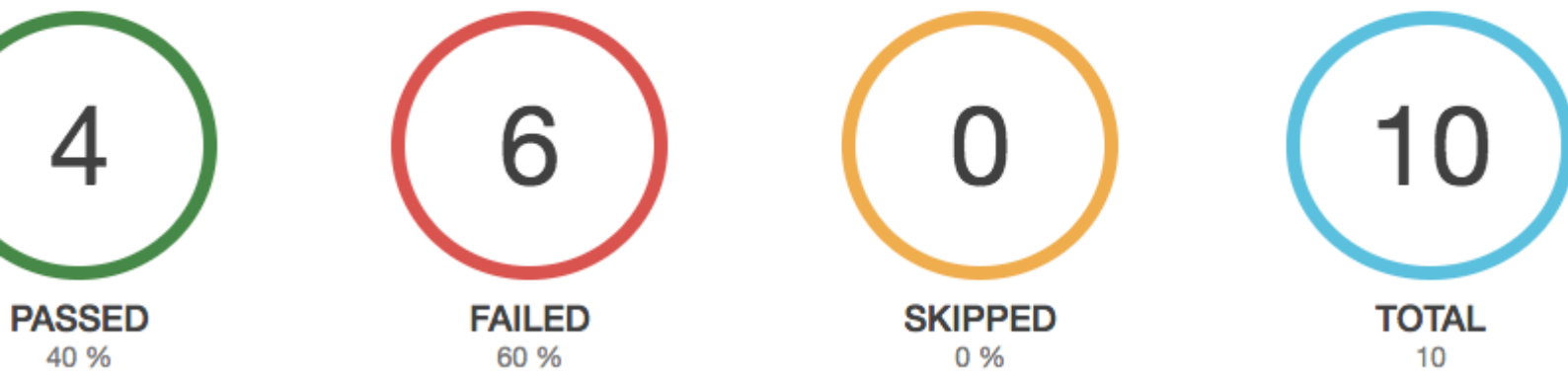
PASSED
40 %



FAILED
60 %



SKIPPED
0 %



When a test case is failing for some reason exception and failure information will be provided.

Chapter 6. Links & Further reading

- The [Citrus reference manual](#) gives you a detailed description of all Citrus features.
- Examples:
 - [Sample projects](#)
 - [Devoxx '15 sample project](#) with Microservices and Docker
 - [JavaLand '16 sample project](#) with Arquillian
- [ChangeLog](#) shows the release history.
- [Contributing](#) explains how you can contribute to this project. Pull requests are highly appreciated!