Citrus Administration UI

Christoph Deppisch

Version 1.0.2, 2017-12-18

citrus-admin

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 considered stable but 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 fo 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 weather 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/1.0.2/citrus-admin-web-1.0.2-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.

Chapter 3. Setup

The administration UI provides several settings and customization parameters that describe how the UI should behave. You can set those parameters as system properties or environment variables on server startup in order to tell the UI how to load projects for instance.

The UI is able to open projects in order to view and edit your project configuration and test cases. Usually the projects are located somewhere on your local machine so the UI can open those projects from local file system. In addition to that the UI is able to load project sources from git repositories. As an alternative you can create completely new projects from Maven archetypes. The following sections walk through all these options in more detail.

3.1. New project

The admin UI is able to create completely new projects. The project sources are saved to your local file system in the admin UI working directory. The working directory is settable via system property or environment variable (see system configuration). By default this is the user home directory on your local system.

3.1.1. Git clone

First of all you can load Citrus project sources from git repositories. Just specify the git repository URL in combination with branch settings and optional sub module names and the UI will load the project sources to your local file system before opening that project.

Repository URL

master

/

Load project   Cancel

Repository URL

master

/

Load project   Cancel

The admin UI performs a git clone command (if the git binaries are available). As a fallback the UI will load the zipped project via Http and unpack the sources to the local file system. Once again the new project sources are saved to the admin UI working directory which is settable via system properties and environment variables (see system configuration).

3.1.2. Maven archetype

The second approach for creating new projects is to generate the project sources from a Maven archetype. Citrus provides several archetypes on Maven central (search.maven.org). Just fill out the archetype forms and enter the Maven archetype coordinates (**groupId**, **artifactId** and **version**) that should be used.

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]

In addition to that you should set new project Maven coordinates for the project that is about to be created on your local file system. As usual these are Maven **groupId**, **archetypeId** and **version**. As a result the admin UI automatically loads the Maven archetype and generates the project sources in the admin UI working directory. The newly create project is then automatically opened in the UI.

3.2. Open project

Of course you can open existing Citrus projects that are located on your local file system. When the administration UI is started without any further options you need to open a Citrus 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 property or environment variable at startup:

```
java -Dcitrus.admin.project.home=/Users/myaccount/path/tp/citrus/project/home -jar
citrus-admin-web-1.0.2.war
```

When pre selecting a project home the project is opened automatically and the project dashboard is displayed.

The project home should point to a project root directory on your local file system that contains the Citrus project sources. You can specify the complete path manually or pick the home directory via the file browser.

project home

ct Citrus project home directory and confirm.

ons

ts
ds

emy
ne
s
quillian-extension-spring
ticles
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

ject home directory and confirm.

sion-spring

ps
nages
s

otation-config
ery
ary
okstore

put
project

Select   **Close**

Once you have specified the project home you are ready to hit the **Open** button. Citrus will read the project information and open the project dashboard. The administration UI is looking for several things in your project in order to gain information about the project. The files scanned are:

Table 1. Project files

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

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

In case one of these mentioned files and directories is not present in your project sources the UI will not open the project due to invalid project sources. Sometimes projects use different file locations and project source file names for some reason. Fortunately you can customize thee folder path and file name settings on admin UI startup.

When you are using a different project layout than expected there are different approaches to customizing these 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-1.0.2.war
```

As you can see we can customize the project file paths that are scanned by the UI. This enables us to open projects with different project file locations. Please refer to the complete list of available system properties in chapter system configuration.

The exact same parameters as seen in the system properties are also available when set as environment variables. The admin UI will automatically read those environment settings on startup. So you can adjust the project file locations using environment variables, too. Please see the complete list of available environment settings in chapter system configuration.

3.3. System configuration

The administration UI is a web application that is started as standalone Spring Boot application or as a deployment in a Java web application server. The admin UI application uses some general settings and customization parameters that describe how the UI should behave. You can set those parameters as system properties or environment variables on server startup in order to tell the UI how to load projects for instance.

3.3.1. System properties

You can set the following system properties in order to customize the admin Ui application:

Table 2. System properties

Property	Description
server.port	Web server port
citrus.admin.project.home	Preselect project on startup

Property	Description
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.project.repository.branch	Git project branch (default: master)
citrus.admin.project.repository.module	Module name (directory) representing the sub module in a Maven multi module repository (default: not set)
citrus.admin.maven.archetype.coordinates	Maven archetype coordinates (groupId:artifactId:version) to auto generate new project on startup (default: not set)
citrus.admin.maven.project.coordinates	Project coordinates for new project generated from archetype (default: com.consol.citrus:citrus-project:1.0.0)
citrus.admin.maven.project.package	Package name for newly generated project (default: com.consol.citrus)
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**)
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)

These properties are specified in the CLI command when starting the application. For instance

```
java -Dcitrus.admin.project.home=/Users/myaccount/path/tp/citrus/project/home -jar
citrus-admin-web-1.0.2.war
```

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

```
java -Dserver.port=8181 -jar citrus-admin-web-1.0.2.war
```

The system properties are automatically identified in the admin UI web application during startup.

3.3.2. Environment variables

The exact same properties that we have seen in the previous system properties section 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_PROJECT_REPOSITORY_BRANCH	Git project branch (default: master)
CITRUS_ADMIN_PROJECT_REPOSITORY_MODULE	Module name (directory) representing the sub module in a Maven multi module repository (default: not set)
CITRUS_ADMIN_MAVEN_ARCHETYPE_COORDINATES	Maven archetype coordinates (groupId:artifactId:version) to auto generate new project on startup (default: not set)
CITRUS_ADMIN_MAVEN_PROJECT_COORDINATES	Project coordinates for new project generated from archetype (default: com.consol.citrus:citrus-project:1.0.0)
CITRUS_ADMIN_MAVEN_PROJECT_PACKAGE	Package name for newly generated project (default: com.consol.citrus)
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**)

The environment settings are very useful when running the admin UI as part of a Docker container infrastructure. Also when running the UI in Kubernetes the use of environment settings is very comfortable as these settings are well suited for configuring Docker related container instances. Read more about that in chapter docker.

3.3.3. Json project settings

Every time the UI has opened a Citrus project settings are stored to the general project settings file located in the root project folder. This is done because the next time you open that project the admin UI should use the exact same project settings as before. The project settings file is stored in the project root folder and is called **citrus-project.json**.

As already mentioned the administration UI creates this project settings file automatically in case it is not present. So when you open a Citrus project for the first time this file is created automatically. Following from that this project settings file should be part of your code versioning platform in order to save settings related to that project. When the project is reopened the UI will read project settings from that file in order to make sure that the project is loaded with the exact same settings as in the past. So you can also adjust this file manually in order to manipulate the way the UI is opening your project. Of course you can also create this file manually prior to opening the project with the admin UI in order to set custom directories and settings from the very beginning.

The setting file uses Json data format and looks like this:

```json
{
  "name" : "citrus-sample-project",
  "description" : "",
  "version" : "1.0.0",
  "settings" : {
    "basePackage" : "com.consol.citrus.samples",
    "citrusVersion" : "2.7.2",
    "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 always review the project settings when looking at this file.

Chapter 4. Project

The basic objective of the administration UI is to manage your Citrus projects. The UI is able to open projects in order to view and edit your project configuration and test cases. Usually the projects are located somewhere on your local machine so the UI can open those projects from local file system. In addition to that the UI is able to load project sources from git repositories. As an alternative you can create completely new projects from Maven archetypes. The following sections walk through all these options in more detail.

4.1. 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
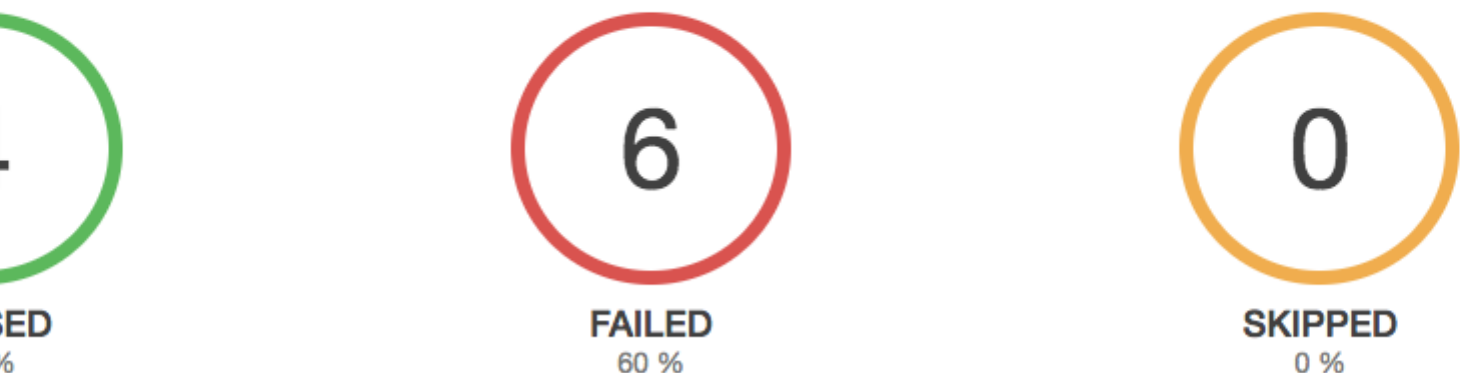
-sample-incident

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

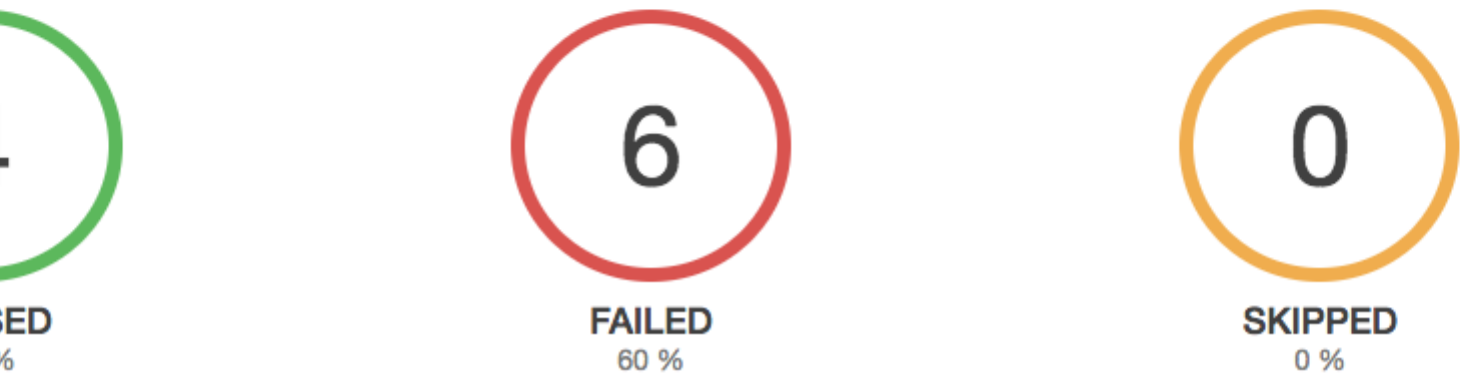consol.citrus.samples

ts 06/26/2017

ED
%

FAILED
60 %

0

SKIPPED
0 %

citrus-sample-incident

-sample-incident

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

consol.citrus.samples

ts 06/26/2017

6

FAILED
60 %

0

SKIPPED
0 %

The project dashboard is a good starting point to discover your project with all projects settings and tests.

4.2. 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 for now.

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:

```json
{
    "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
      }
    }
}
```

This project Json file is automatically created whenever the admin UI opens a project on your local file system. This file should be part of your coder versioning platform in order to save settings related to that project. When the project is reopened the UI will read project settings from that file in order to make sure that the project is loaded with the exact same settings as in the past. So you can also adjust this file manually in order to manipulate the way the UI is opening your project.

4.2.1. 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.

4.2.2. Build configuration

The administration web UI is able to execute tests. This test execution is done by calling the Maven build lifecylce 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

4.2.3. 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.

gs

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

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

eeds. Citrus provides separate modules for message transports and exchanged data. By selecting the modules below Citrus wil

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!

4.2.4. 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.
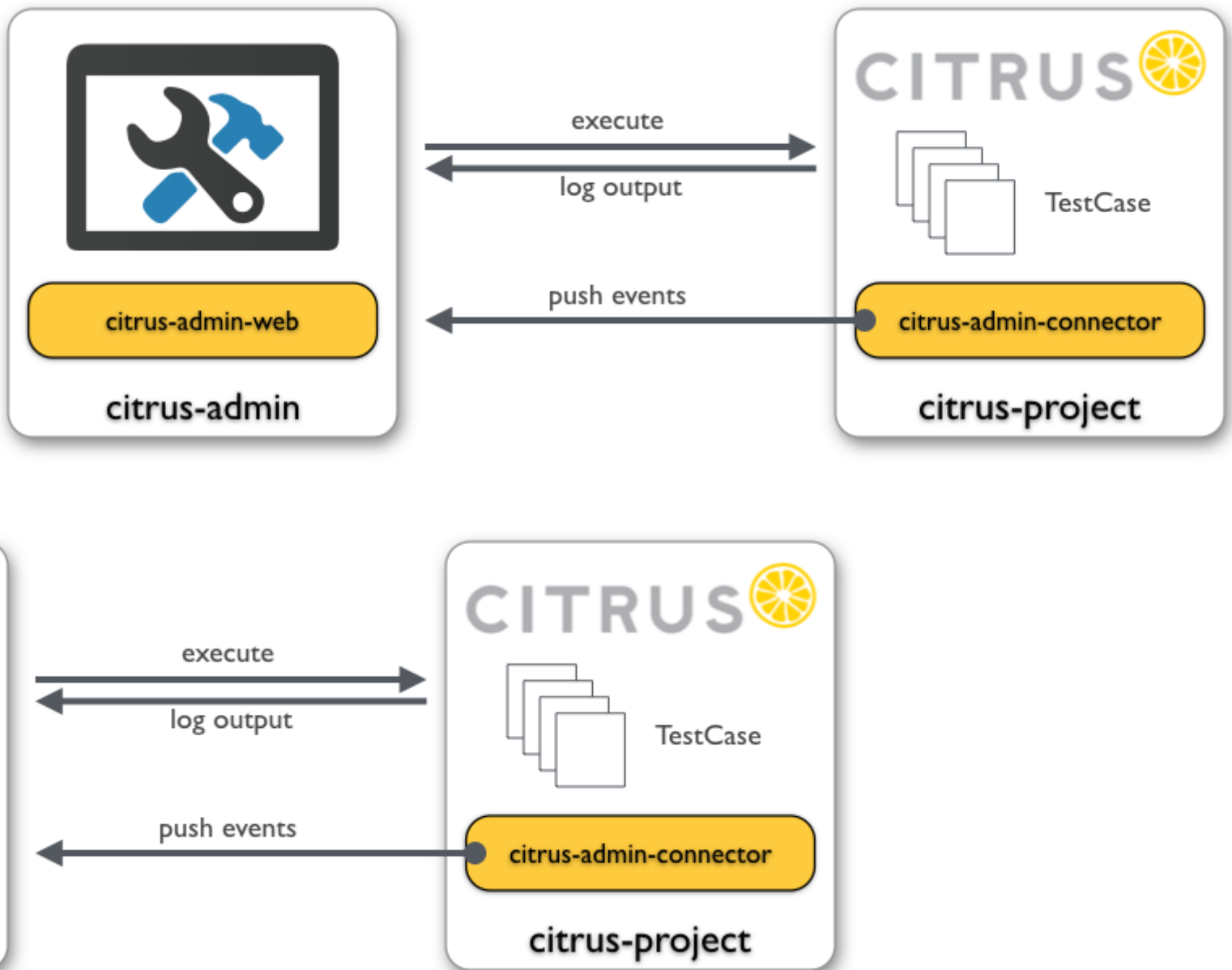
The connector library is available from Maven central and is simply added as dependency to the target project. You can use the admin UI settings page for automatically adding this little helper to the target project. The automated connector setting will place the new Maven dependency to the project POM 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 5. 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.
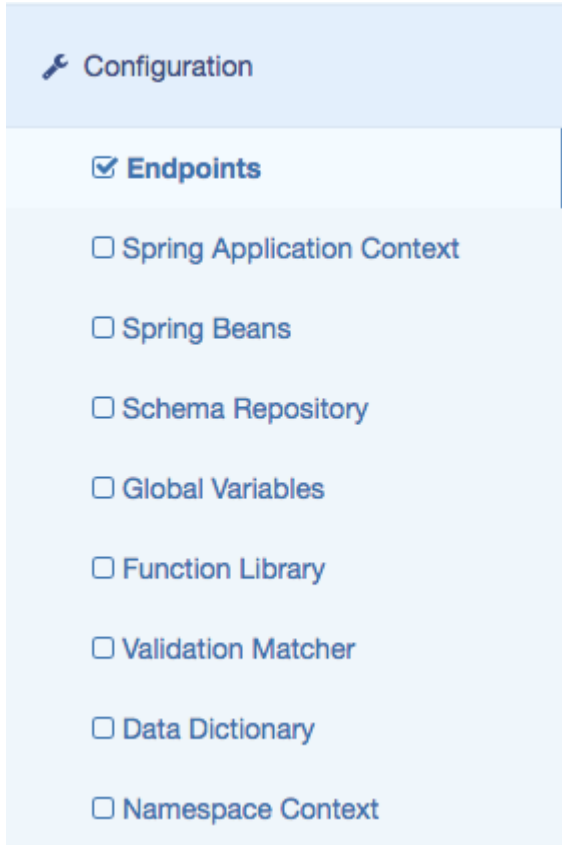
5.1. Endpoint configuration

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
messages.

	Properties
…int	**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

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

	Properties
...int	**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

	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.
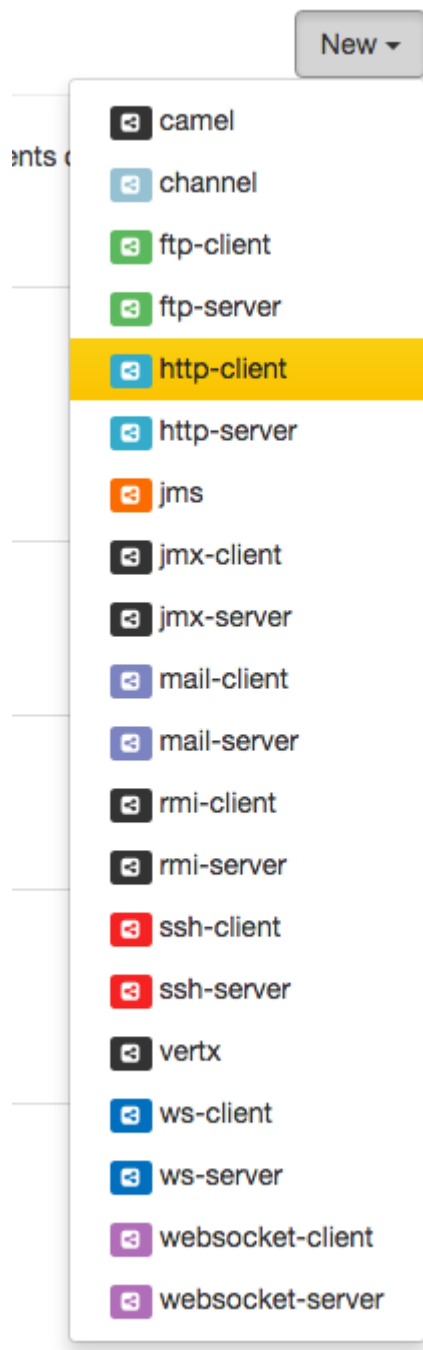
New ▾

📦 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

ents

ttp-client

ST

bagateError

0

00

ve Close

ttp-client

ST

bagateError

0

00

ve | Close

	Name is required

	RequestUrl is required

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 an existing endpoint

component and you will see the HTML form with all the settings to this endpoint.

client

entHttpClient

/localhost:18001/incident/IncidentManager/v1

Interceptors

sException

Close

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.

5.2. Spring application context

Citrus uses the Spring framework for dependency injection and component configuration. Usually a Citrus project comes with a set of Spring configuration files that define Citrus components and

other project related settings. The admin UI scanns the project sources for typical Spring configuration files and displays all components (Spring beans) in that configuraiton.

You can review the beans with all properties and you can edit the Spring configuration files.

on context

pring application context files. See the following context files that are known in your project.

```
'?>
ework.org/schema/beans"
/2001/XMLSchema-instance"
rusframework.org/schema/config"
.citrusframework.org/schema/jms/config"
citrusframework.org/schema/ws/config"
w.citrusframework.org/schema/http/config"
ringframework.org/schema/context"
gframework.org/schema/util"
framework.org/schema/oxm"

rg/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
rg/schema/context http://www.springframework.org/schema/context/spring-context.xsd
rg/schema/util http://www.springframework.org/schema/util/spring-util.xsd
rg/schema/config http://www.citrusframework.org/schema/config/citrus-config.xsd
rg/schema/jms/config http://www.citrusframework.org/schema/jms/config/citrus-jms-config.xsd
rg/schema/ws/config http://www.citrusframework.org/schema/ws/config/citrus-ws-config.xsd
rg/schema/http/config http://www.citrusframework.org/schema/http/config/citrus-http-config.xsd
rg/schema/oxm http://www.springframework.org/schema/oxm/spring-oxm.xsd">

ation="classpath:citrus.properties"/>


emaRepository">

ocation="classpath:schemas/soap-envelope-1-1.xsd"/>
nagerWsdl" location="classpath:com/consol/citrus/samples/incident/schema/IncidentManager.wsdl"/>
viceXsd" location="classpath:com/consol/citrus/samples/incident/schema/NetworkService.xsd"/>
ServiceXsd" location="classpath:com/consol/citrus/samples/incident/schema/FieldForceService.xsd"/>
Wsdl" location="classpath:com/consol/citrus/samples/incident/schema/SmsGateway.wsdl"/>




name" value="Citrus IncidentManager sample"/>
```

pring application context files. See the following context files that are known in your project.

```
"?>
ework.org/schema/beans"
/2001/XMLSchema-instance"
rusframework.org/schema/config"
.citrusframework.org/schema/jms/config"
citrusframework.org/schema/ws/config"
w.citrusframework.org/schema/http/config"
ringframework.org/schema/context"
gframework.org/schema/util"
framework.org/schema/oxm"

rg/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
rg/schema/context http://www.springframework.org/schema/context/spring-context.xsd
rg/schema/util http://www.springframework.org/schema/util/spring-util.xsd
rg/schema/config http://www.citrusframework.org/schema/config/citrus-config.xsd
rg/schema/jms/config http://www.citrusframework.org/schema/jms/config/citrus-jms-config.xsd
rg/schema/ws/config http://www.citrusframework.org/schema/ws/config/citrus-ws-config.xsd
rg/schema/http/config http://www.citrusframework.org/schema/http/config/citrus-http-config.xsd
rg/schema/oxm http://www.springframework.org/schema/oxm/spring-oxm.xsd">

ation="classpath:citrus.properties"/>


emaRepository">

ocation="classpath:schemas/soap-envelope-1-1.xsd"/>
nagerWsdl" location="classpath:com/consol/citrus/samples/incident/schema/IncidentManager.wsdl"/>
viceXsd" location="classpath:com/consol/citrus/samples/incident/schema/NetworkService.xsd"/>
ServiceXsd" location="classpath:com/consol/citrus/samples/incident/schema/FieldForceService.xsd"/>
Wsdl" location="classpath:com/consol/citrus/samples/incident/schema/SmsGateway.wsdl"/>


name" value="Citrus IncidentManager sample"/>
```

t files. See the following context files that are known in your project.

```
s"
nce"
ma/config"
schema/jms/config"
chema/ws/config"
/schema/http/config"
ema/context"
/util"
oxm"

//www.springframework.org/schema/beans/spring-beans.xsd
p://www.springframework.org/schema/context/spring-context.xsd
/www.springframework.org/schema/util/spring-util.xsd
://www.citrusframework.org/schema/config/citrus-config.xsd
http://www.citrusframework.org/schema/jms/config/citrus-jms-config.xsd
ttp://www.citrusframework.org/schema/ws/config/citrus-ws-config.xsd
  http://www.citrusframework.org/schema/http/config/citrus-http-config.xsd
www.springframework.org/schema/oxm/spring-oxm.xsd">

us.properties"/>


hemas/soap-envelope-1-1.xsd"/>
classpath:com/consol/citrus/samples/incident/schema/IncidentManager.wsdl"/>
asspath:com/consol/citrus/samples/incident/schema/NetworkService.xsd"/>
"classpath:com/consol/citrus/samples/incident/schema/FieldForceService.xsd"/>
path:com/consol/citrus/samples/incident/schema/SmsGateway.wsdl"/>



ncidentManager sample"/>
```

to the Citrus components and the project in general. On this page you see all bean configurations in your project.

Type	Properties
type: ActiveMQConnectionFactory **package**: org.apache.activemq	**brokerURL**: tcp://localhost:61616
type: LoggingClientInterceptor **package**: com.consol.citrus.ws.interceptor	
type: MessageTracingTestListener **package**: com.consol.citrus.report	
type: SaajSoapMessageFactory **package**: org.springframework.ws.soap.saaj	
type: SimpleSoapAttachmentValidator **package**: com.consol.citrus.ws.validation	
type: SimpleSoapFaultValidator **package**: com.consol.citrus.ws.validation	
type: SoapJmsMessageConverter **package**: com.consol.citrus.jms.message	
type: LoggingHandlerInterceptor **package**: com.consol.citrus.http.interceptor	

to the Citrus components and the project in general. On this page you see all bean configurations in your project.

Type	Properties
type: ActiveMQConnectionFactory **package:** org.apache.activemq	**brokerURL:** tcp://localhost:61616
type: LoggingClientInterceptor **package:** com.consol.citrus.ws.interceptor	
type: MessageTracingTestListener **package:** com.consol.citrus.report	
type: SaajSoapMessageFactory **package:** org.springframework.ws.soap.saaj	
type: SimpleSoapAttachmentValidator **package:** com.consol.citrus.ws.validation	
type: SimpleSoapFaultValidator **package:** com.consol.citrus.ws.validation	
type: SoapJmsMessageConverter **package:** com.consol.citrus.jms.message	
type: LoggingHandlerInterceptor **package:** com.consol.citrus.http.interceptor	

t in general. On this page you see all bean configurations in your project.

	Properties	Actions
	brokerURL: tcp://localhost:61616	☑ | ✖
erceptor		☑ | ✖
		☑ | ✖
.soap.saaj		☑ | ✖
ator lidation		☑ | ✖
lidation		☑ | ✖
essage		☑ | ✖
nterceptor		☑ | ✖

5.3. Global Variables

Test variables represent a common concept in Citrus. Variables define reusable identifiers and values that are referenced multiple times in a test case. The global variables are visible to all Citrus test cases and represent static constant values valid for the whole project. The global variables are editable via HTML forms on the configuration page:

es

t are valid for all test cases. You should put in here constant values that are used across all test cases. Inside the test case you

Value
Citrus IncidentManager sample

es

t are valid for all test cases. You should put in here constant values that are used across all test cases. Inside the test case you

Value
Citrus IncidentManager sample

t in here constant values that are used across all test cases. Inside the test case you can use those global variables with the

ager sample

The variables are saved to the Spring bean configuration as Citrus **global-variables** component:

```
<citrus:global-variables>
    <citrus:variable name="projectName" value="Citrus Integration Testing"/>
    <citrus:variable name="userName" value="TestUser"/>
</citrus:global-variables>
```

5.4. Schema Repositories

Every time Citrus receives a message over some kind of messaging transport the validation mechanism will try to validate the syntax of the message payload with some schema rules. When using XML payloads these are XSD or WSDL files that define the syntax rules. In Citrus you can define schema repositories that hold one to many XML schema definition files. The Citrus XML message validators consult these schema repositories during message validation for matching schema definitions.

The schema repositories and schema definition files are managed on configuration page.

itories

ber schema definition checks are required in order to ensure syntax and interface contract rules. In Citrus we define all our sche
tomatically during a test run and incoming XML messages are validated accordingly.

s. See all available repositories listed below:

	Schemas
	schemas: soapEnv | incidentManagerWsdl | networkServiceXsd | fieldForceServiceXsd | smsGatewayWsdl |

reused in multiple schema repositories. Schema repositories do simply reference the schemas defined in global scope. Global s
define the schema as private member of the schema repository itself.

	Location
	classpath:schemas/soap-envelope-1-1.xsd
	classpath:com/consol/citrus/samples/incident/schema/IncidentManager.wsdl
	classpath:com/consol/citrus/samples/incident/schema/NetworkService.xsd
	classpath:com/consol/citrus/samples/incident/schema/FieldForceService.xsd
	classpath:com/consol/citrus/samples/incident/schema/SmsGateway.wsdl

per schema definition checks are required in order to ensure syntax and interface contract rules. In Citrus we define all our sche
utomatically during a test run and incoming XML messages are validated accordingly.

s. See all available repositories listed below:

	Schemas
	schemas: soapEnv	incidentManagerWsdl	networkServiceXsd	fieldForceServiceXsd	smsGatewayWsdl	

reused in multiple schema repositories. Schema repositories do simply reference the schemas defined in global scope. Global s
define the schema as private member of the schema repository itself.

	Location
	classpath:schemas/soap-envelope-1-1.xsd
	classpath:com/consol/citrus/samples/incident/schema/IncidentManager.wsdl
	classpath:com/consol/citrus/samples/incident/schema/NetworkService.xsd
	classpath:com/consol/citrus/samples/incident/schema/FieldForceService.xsd
	classpath:com/consol/citrus/samples/incident/schema/SmsGateway.wsdl

are required in order to ensure syntax and interface contract rules. In Citrus we define all our schema definitions in schema-rep
and incoming XML messages are validated accordingly.

es listed below:

soapEnv	incidentManagerWsdl	networkServiceXsd	fieldForceServiceXsd	smsGatewayWsdl	

positories. Schema repositories do simply reference the schemas defined in global scope. Global scoped schemas should alwa
member of the schema repository itself.

chemas/soap-envelope-1-1.xsd
om/consol/citrus/samples/incident/schema/IncidentManager.wsdl
om/consol/citrus/samples/incident/schema/NetworkService.xsd
om/consol/citrus/samples/incident/schema/FieldForceService.xsd
om/consol/citrus/samples/incident/schema/SmsGateway.wsdl

The configuration made here is saved to the Spring bean configuration files as Citrus schema components.

5.5. Namespace Context

When using Xpath expressions in Citrus XML namespaces may be required to identify elements and attributes in XML payloads. The XML namespaces used in Xpath are identified with a prefix that evaluates to a target namespace value. You can define all namespaces and prefix values globally using the namespace context configuration.

[Namespace context]

The namespace context configuration is save as Citrus Spring bean component in the application context.

```
<citrus:namespace-context>
    <citrus:namespace prefix="hello"
uri="http://citrusframework.org/schemas/samples/HelloService.xsd"/>
</citrus:namespace-context>
```

5.6. Data Dictionaries

Data dictionaries help to define reusable expressions that manipulate message payloads before a message is sent or received. Usually these are Xpath or JsonPath expressions that overwrite values in message payloads throughout the whole project. Each message payload will consult the data dictionary for translation of values before being sent or received.

This way you can add centralized expressions that overwrite message payload elements in multiple test cases.

The data dictionary configuration is done on the configuration pages.

es

ppings that are valid to all test cases and all messages sent and received across Citrus. The data dictionary is asked for translat y the respective dictionary value is set in the message. This is a way of centralized message adjustments in Citrus. The mapping paths. Every time a message is sent or received the data dictionary matching the mapping key will apply to the message value es. See all available dictionaries listed below:

	Mappings
	📄 /root/person/name()
	📄 /root/person/age()

ppings that are valid to all test cases and all messages sent and received across Citrus. The data dictionary is asked for translat
y the respective dictionary value is set in the message. This is a way of centralized message adjustments in Citrus. The mapping
paths. Every time a message is sent or received the data dictionary matching the mapping key will apply to the message value
es. See all available dictionaries listed below:

	Mappings
	📄 /root/person/name() 📄 /root/person/age()

s and all messages sent and received across Citrus. The data dictionary is asked for translation of message elements. If the
set in the message. This is a way of centralized message adjustments in Citrus. The mapping keys in a data dictionary can be X
nt or received the data dictionary matching the mapping key will apply to the message value before validation takes place.
ted below:

Mappings
📄 /root/person/name() 📄 /root/person/age()

5.7. Functions

Functions apply on message payloads and test variables for dynamic value generation. The functions can be used in multiple places throughout the Citrus framework (e.g. payload, header, variables, etc.) Each function is provided with parameters and generates a String value as outcome. This way the user can place functions in order to generate more dynamic test data for instance. There are lots of predefined functions ready for usage in Citrus (e.g. citrus:randomUUID(), citrus:currentDate() and many more).

In addition to that you can create custom functions that implement very specific algorithms matching your very specific project needs. These functions and its implementations are configured here:

er to provide more dynamic values such as the current date, timestamps, random numbers and so on. The functions have a nar
you can easily write your own functions and put them in a custom library in order to extend the framework capabilities.

raries. See all available libraries listed below:

Prefix	Members
custom	📄 **customFunction()**

New

he current date, timestamps, random numbers and so on. The functions have a name and are grouped in a function library.
put them in a custom library in order to extend the framework capabilities.

:

Members	
📄 **customFunction()**	✖

5.8. Validation Matcher

Similar to adding custom functions in Citrus you can also add custom validation matcher

implementations. The matcher helpers do implement specific validation logic that applies to message validation when receiving messages in Citrus. There are lots of predefined matchers available in Citrus (e.g. @contains()@, @equalsIgnoreCase()@ and many more).

Custom validation matchers are configurable via HTML forms on the configuration pages.

tcher Libraries

is able to consult the service of validation matcher implementations. These validation matcher bring more flexibility to a validatio
ion matcher help to get more powerful validation capabilities. Citrus offers a default list of validation matcher implementations s
tion matcher and put it to some custom validation matcher library.

natcher libraries. See all available libraries listed below:

	Prefix	Matcher
	custom	📄 customMatcher()

New

atcher implementations. These validation matcher bring more flexibility to a validation operation. When comparing a received
ation capabilities. Citrus offers a default list of validation matcher implementations such as equalsIgnoreCase or isNumber. Of
dation matcher library.

sted below:

refix	Matcher	
ustom	📄 **customMatcher()**	✖

Chapter 6. Tests
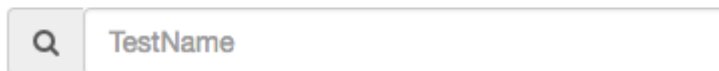
Managing all your tests is a basic thing to do with the administration UI. You are able to search for tests, open tests and execute tests with the UI.

You can do the following things regarding test management

- Search tests
- List tests
- Open tests
- Execute tests
- Test designer
- Reporting
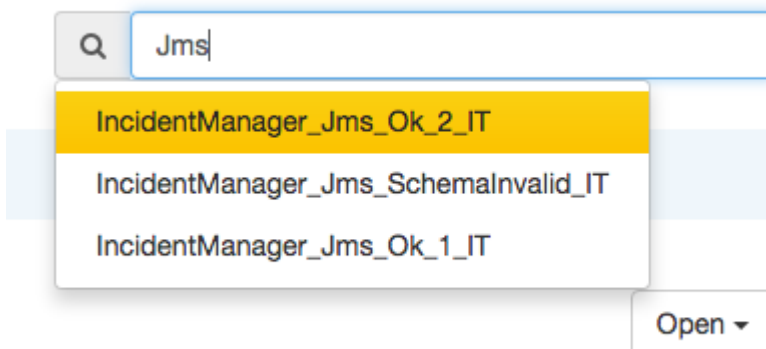
6.1. Search tests

Searching for test cases is very simple. Open the tests page and you will se 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.

6.2. 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.

est cases

TestName

	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

est 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
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

6.3. 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.

rus integration test cases

rus integration test cases

🔍 TestName

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

IncidentManager_Http_Ok_3_IT	com.consol.c
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	

Q TestName

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

IncidentManager_Http_Ok_3_IT	com.consol.c
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	

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

cident **Last Result:** SUCCESS

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

```
;

usXmlTest;
estNGCitrusTest;



extends AbstractTestNGCitrusTest {


_Http_Ok_1_IT")
Ok_1() {
```

```
;

usXmlTest;
estNGCitrusTest;




extends AbstractTestNGCitrusTest {


_Http_Ok_1_IT")
0k_1() {
```

```
:;

usXmlTest;
estNGCitrusTest;



                    extends AbstractTestNGCitrusTest {


r_Http_Ok_1_IT")
Ok_1() {
```

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

```
mework.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"
ringframework.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
ork.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
ork.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
ork.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>

rus:randomUUID()"/>
itrus:randomNumber(6)"/>

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>

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

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

```
mework.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"
ringframework.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
ork.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
ork.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
ork.org/schema/testcase http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">

1_IT">


te>

/last-updated-by>
0</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>


rus:randomUUID()"/>
itrus:randomNumber(6)"/>



 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>

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


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

```
ework.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"
ringframework.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
rk.org/schema/jms/testcase http://www.citrusframework.org/schema/jms/testcase/citrus-jms-testcase.xsd
rk.org/schema/ws/testcase http://www.citrusframework.org/schema/ws/testcase/citrus-ws-testcase.xsd
rk.org/schema/testcase http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">

1_IT">

te>

/last-updated-by>
0</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>

rus:randomUUID()"/>
itrus:randomNumber(6)"/>

 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>

: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.

6.4. 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.

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 ---

l2java (apache-cxf-generate) @ citrus-sample-incident ---

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


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

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 ---

l2java (apache-cxf-generate) @ citrus-sample-incident ---

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


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

Run

ples.incident **Last Result:** SUCCESS   71

100%

SUCCESS

```
_____
_____

-incident ---
incident/target

t ---

itrus-sample-incident ---

itrus-sample-incident ---


us-sample-incident ---

trus-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.

ing="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>

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>

```
s:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

hema/samples/IncidentManager/v1">

db10</im:ticketId>
l>

 software!</im:description>

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 non of these approaches is working for you the admin UI will not display exchanged messages after the test run.

6.5. 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.

6.6. 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

st test results

r the active

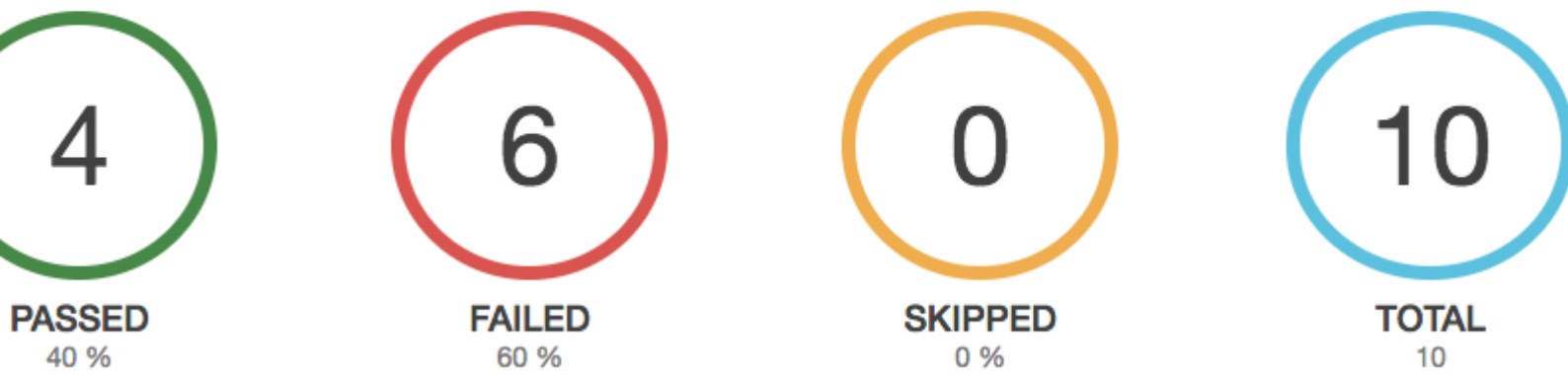| 4	6	0	10
PASSED	**FAILED**	**SKIPPED**	**TOTAL**
40 %	60 %	0 %	10

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

Chapter 7. Docker support

7.1. Docker image

The administration UI is available as Docker image (`consol/citrus-admin:latest`). You can pull the image and link it to your local Citrus project:

```
docker run -d -p 8080:8080 -v $PWD:/maven -e CITRUS_ADMIN_PROJECT_HOME=/maven
consol/citrus-admin:latest
```

The command above loads the Docker image and runs a new Citrus web UI container. The container is provided with a volume mount that makes the current directory accessible from within the container. This current directory is then used as project home so the admin UI will automatically open the Citrus project from that directory. Once the container is running you can point your local browser to link:[http://localhost:8080] in order to access the web UI.

The `CITRUS_ADMIN_PROJECT_HOME` environment setting is optional and is used to automatically open a project on container startup. You can leave out this setting in order to select a project folder in your mounted working directory when starting the web UI.

In case you do not have a Citrus project ready yet, the admin UI can also create a new project for you. It is possible to run a Maven archetype on container startup that creates a complete new project for you. You can set the Maven archetype coordinates (**groupId**, **artifactId**, **version**) as environment variables when running the container.

```
docker run -d -p 8080:8080 -v $PWD:/maven -e
CITRUS_ADMIN_MAVEN_ARCHETYPE_COORDINATES=com.consol.citrus.mvn:citrus-quickstart:2.7.2
consol/citrus-admin:latest
```

The UI will load the Maven archetype and create the project sources when the container is started. The new project gets its Maven coordinates from another environment setting:

```
-e CITRUS_ADMIN_MAVEN_PROJECT_COORDINATES=com.consol.citrus:citrus-sample:1.0.0
```

Another way to load a new project on container startup is to specify a git repository URL. The Citrus admin Docker container will then load the project sources from that git repository on startup:

```
docker run -d -p 8080:8080 -v $PWD:/maven -e
CITRUS_ADMIN_PROJECT_REPOSITORY=https://github.com/account/citrus-project.git
consol/citrus-admin:latest
```

The command above will load the project sources from git with URL https://github.com/account/citrus-project.git and open that project afterwards. The git repository of course should hold the Citrus project sources. In case the Citrus project is located in a sub module

in that git repository you can load that sub module by specifying additional environment properties:

```
-e CITRUS_ADMIN_PROJECT_REPOSITORY_MODULE=/integration/citrus-test -e
CITRUS_ADMIN_PROJECT_REPOSITORY_BRANCH=bugfix
```

With these options we are able to start the Docker image as container with special customizations via environment settings. Please refer to the complete list of supported environment variables in chapter system configuration.

7.2. Use Docker Maven plugin

The citrus-admin Docker image works great with the Fabric8 Docker Maven plugin (https://dmp.fabric8.io/). You can add the plugin configuration as follows in your Maven POM:

```xml
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <configuration>
      <verbose>true</verbose>
      <images>
        <image>
          <alias>citrus-admin</alias>
          <name>consol/citrus-admin:latest</name>
          <run>
            <namingStrategy>alias</namingStrategy>
            <ports>
              <port>8080:8080</port>
            </ports>
            <volumes>
              <from>
                <image>application</image>
              </from>
            </volumes>
            <env>
              <CITRUS_ADMIN_PROJECT_HOME>/maven</CITRUS_ADMIN_PROJECT_HOME>
            </env>
            <wait>
              <http>
                <url>http://localhost:8080/setup</url>
                <method>GET</method>
                <status>200</status>
              </http>
              <time>60000</time>
              <shutdown>500</shutdown>
            </wait>
            <log>
              <enabled>true</enabled>
              <color>green</color>
            </log>
          </run>
        </image>
        <image>
          <alias>application</alias>
          <name>application:${project.version}</name>
          <build>
            <assembly>
              <descriptorRef>project</descriptorRef>
            </assembly>
          </build>
        </image>
      </images>
    </configuration>
</plugin>
```

Now you can build the images locally with Maven calling

```
mvn docker:build
```

After that you should have a set of new images on your Docker host. You can run these images as Docker container.

```
mvn docker:start
```

7.3. Environment settings

When using Docker images it is good practice to provide environment variables that are able to overwrite general properties in the application in order to adjust the container behavior. The Docker image is able to set several environment properties. For a detailed list of these variables and their meaning in the admin UI web application please refer to chapter setup environment variables.

Chapter 8. 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!