

Project

Table of Contents

New project	1
Git clone	2
Maven archetype	3
Open project	3
Customize settings	7
Dashboard	9
Settings	12
General	13
Build configuration	14
Modules	14
Admin connector	17

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.

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

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

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).

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

ct Citrus project home directory and confirm.

ons

ts

ds

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

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>)

Property	Description
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)

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.

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.

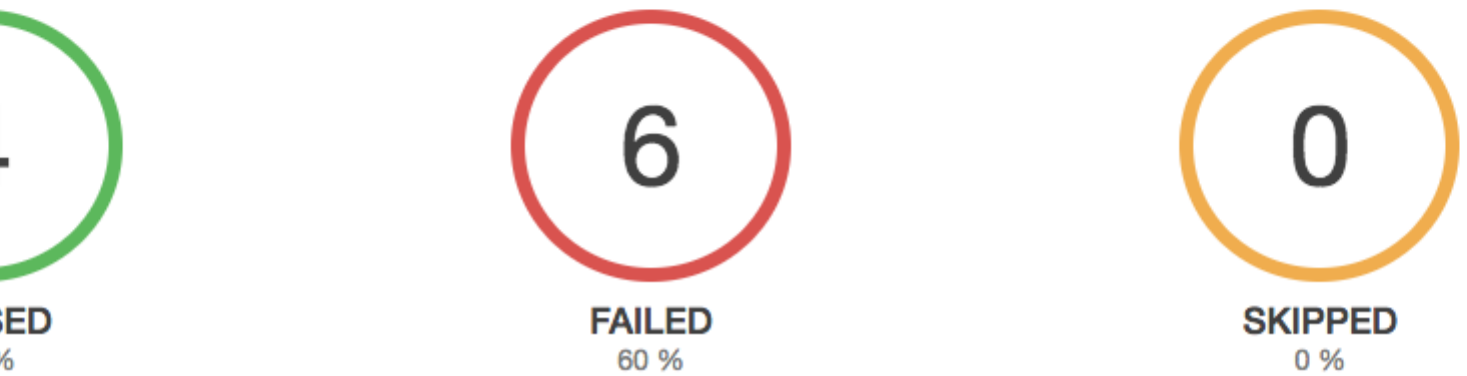
citrus-sample-incident

-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



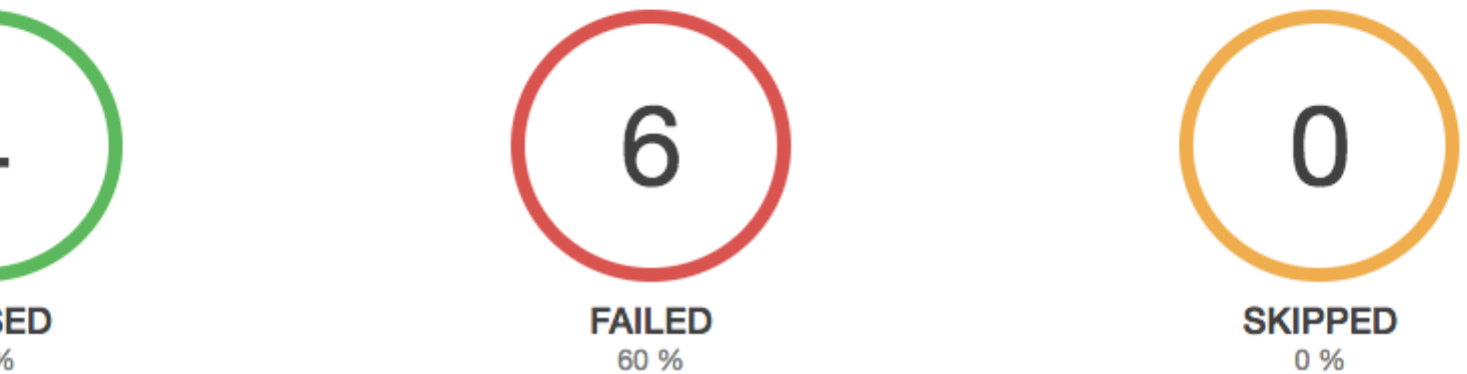
citrus-sample-incident

-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](#).

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
    }
  }
}
```

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.

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

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

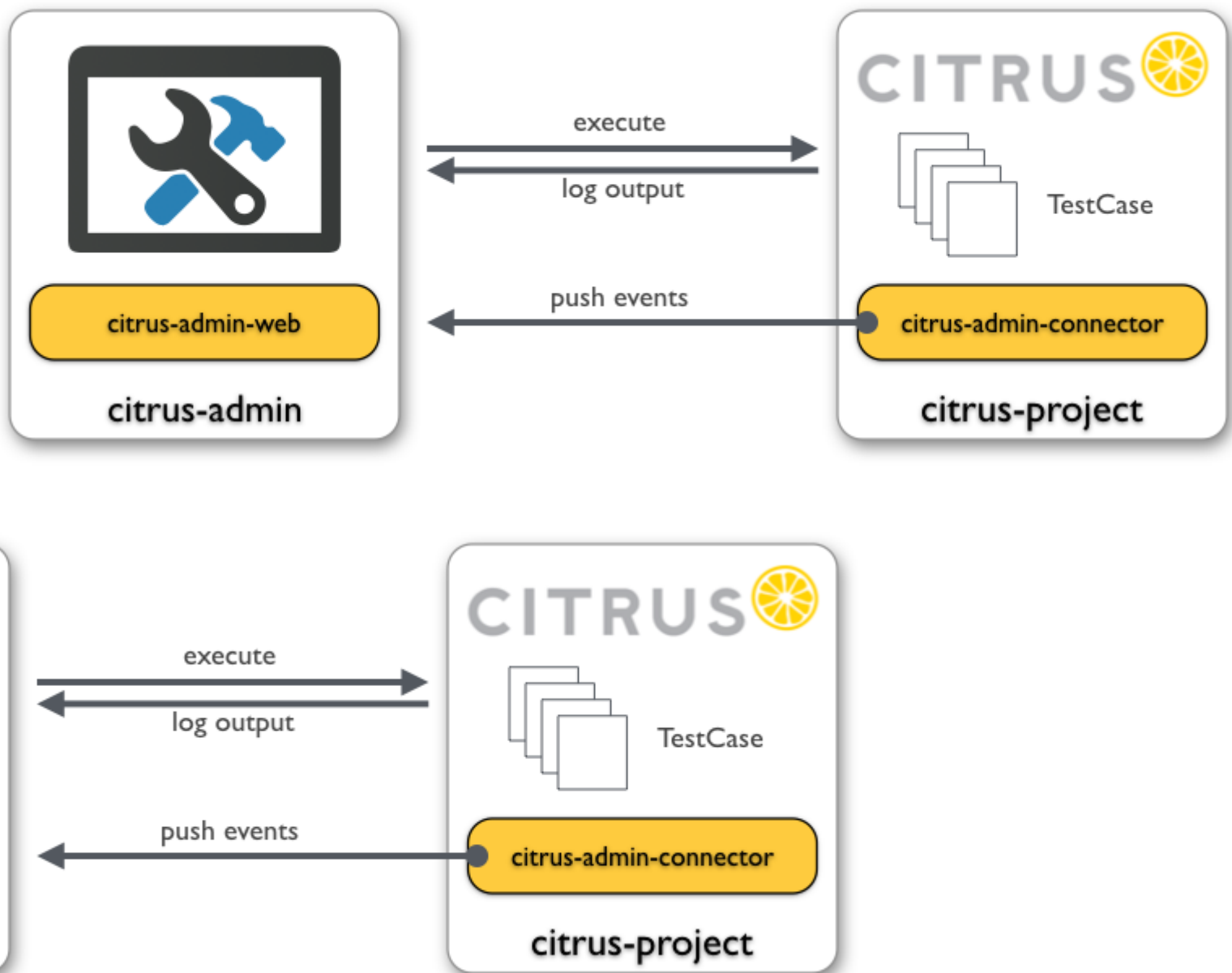
On the modules settings page you see all currently activated modules. And you get a list of available modules that you can add. Just check or uncheck the modules and the administration UI will automatically add/remove Maven dependencies in your project.

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

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!