

Integration with TeamCity (legacy)



These instructions are deprecated!

Xray provides a free plugin for TeamCity. Please use it instead, as described in [Integration with TeamCity](#).

It is easy to setup TeamCity in order to integrate it with Xray.

Since Xray provides a full REST API, you may interact with Xray, for submitting results for example.

- [JUnit example](#)
 - [Run automated tests](#)
 - [Import execution results](#)
- [Cucumber example](#)
 - [Exporting Cucumber features](#)
 - [Run Cucumber scenarios](#)
 - [Import execution results](#)

JUnit example

In this scenario, we want to get visibility of the automated test results from some tests implemented in Java, using the JUnit framework.

This recipe could also be applied for other frameworks such as NUnit or Robot.

CalcTest.java

```
package com.xpand.java;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

public class CalcTest {

    @Before
    public void setUp() throws Exception {

    }

    @After
    public void tearDown() throws Exception {

    }

    @Test
    public void CanAddNumbers()
    {
        assertThat(Calculator.Add(1, 1), is(2));
        assertThat(Calculator.Add(-1, 1), is(0));
    }

    @Test
    public void CanSubtract()
    {
        assertThat(Calculator.Subtract(1, 1), is(0));
        assertThat(Calculator.Subtract(-1, -1), is(0));
        assertThat(Calculator.Subtract(100, 5), is(95));
    }

    @Test
    public void CanMultiply()
    {
        assertThat(Calculator.Multiply(1, 1), is(1));
        assertThat(Calculator.Multiply(-1, -1), is(1));
        assertThat(Calculator.Multiply(100, 5), is(500));
    }

    public void CanDivide()
    {
        assertThat(Calculator.Divide(1, 1), is(1));
        assertThat(Calculator.Divide(-1, -1), is(1));
        assertThat(Calculator.Divide(100, 5), is(20));
    }

    @Test
    public void CanDoStuff()
    {
        assertThat(true, is(true));
    }

}
```

In order to submit those results, we'll just need to invoke the REST API (as detailed in [Import Execution Results - REST](#)).

Run automated tests

Our project is Maven based, therefore the first Build Step compiles and runs the JUnit automated tests.

TC

Projects | ▾

Changes

Agents 1

Build Queue 0

admin | ▾Administration

Administration / <Root project> / java-junit-calc

Run...Actions ▾Build Configuration Home

Build

General Settings

Version Control Settings 1

Build Steps 2

Triggers 1

Failure Conditions

Build Features

Dependencies

Parameters 3

Agent Requirements

Last edited 14 hours ago by admin (view history)

Build Steps

In this section you can configure the sequence of build steps to be executed. Each build step is represented by a build runner and provides integration with a specific build or test tool.

+ Add build step

Reorder build steps

Auto-detect build steps

Build Step	Parameters Description	Edit	
1. Maven	Path to POM: java-junit-calc/pom.xml Goals: clean test Execute: If all previous steps finished successfully	Edit	
2. Import results to Xray	Command Line Custom script: curl -H "Content-Type: multipart/form-da... Execute: Even if some of the previous steps failed	Edit	

Build Step (1 of 2): Maven ▾

Runner type:

Maven

Runs Maven builds

Step name:

Optional, specify to distinguish this build step from other steps.

Goals:

clean test

Space-separated goals to execute.

Path to POM file:

java-junit-calc/pom.xml

The specified path should be relative to the checkout directory.

Code Coverage

Choose coverage runner:

<No coverage>

Show advanced options

Import execution results

In order to submit the results, we'll need to add a Build Step of type "Command Line", where we'll invoke the REST API, submitting the JUnit XML report generated in the previous step.

Build Step (2 of 2): Import results to Xray | ▾

[+ Add build step](#)

Runner type: Command Line ▾
Simple command execution

Step name: Import results to Xray
Optional, specify to distinguish this build step from other steps.

Execute step: Even if some of the previous steps failed ▾
Specify the step execution policy.

Working directory: 📁 🚀
Optional, set if differs from the checkout directory.

Run: Custom script ▾

Custom script: ⚠
Enter build script content:

```
curl -H "Content-Type: multipart/form-data" -u %jira_user%:%jira_password% -F "file=@java-junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTest.xml" "%jira_base_url%/rest/raven/1.0/import/execution/junit?projectKey=CALC&fixVersion=v3.0&revision=1234"
```


A platform-specific script, which will be executed as a .cmd file on Windows or as a shell script in Unix-like environments.

Format stderr output as: warning ⬇
Specify how stderr output is processed.

The complete script content of the "custom script" field above is:

```
curl -H "Content-Type: multipart/form-data" -u %jira_user%:%jira_password% -F "file=@java-junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTest.xml" "%jira_base_url%/rest/raven/1.0/import/execution/junit?projectKey=CALC&fixVersion=v3.0&revision=1234"
```

We're using "curl" utility that comes in Unix based OS'es but you can easily use another tool to make the HTTP request.

Notice that we're using some parameters for storing Jira's base URL along with the credentials to be used in the REST API.

Actually, these parameters can be defined at multiple levels; in our example we defined them at the "Build Configuration" level but they could also have been defined at the project level.

Administration / 📁 <Root project> / 📁 java-junit-calc

📁 Build

[General Settings](#)
[Version Control Settings 1](#)
[Build Steps 2](#)
[Triggers 1](#)
[Failure Conditions](#)
[Build Features](#)
[Dependencies](#)
[Parameters 3](#)
[Agent Requirements](#)

[+ Add new parameter](#)

Configuration Parameters

Configuration parameters are not passed into build, can be used in references only. ⓘ

Name	Value
jira_base_url	http://192.168.56.102
jira_password	*****
jira_user	admin

The parameters can be hidden, such as the password, if you defined them as being of type "Password".

Edit parameter specification

Label:

Custom label to be shown in custom run build dialog instead of parameter name

Description:

Jira password

Description to be shown in custom run build dialog

Display:

Normal

Use 'Hidden' to hide parameter from custom run dialog.
Use 'Prompt' to force custom run dialog with the parameter displayed on every build start.

Read-only:

☐

Make the parameter impossible to override with another value

Type: *

Password

No options are available for chosen type

Save

Cancel

Cucumber example

In this scenario, we are managing the specification of Cucumber Scenarios/Scenario Outline(s) based tests in Jira, as detailed in the "standard workflow" mentioned in [Testing with Cucumber](#)

Then we need to extract this specification from Jira (i.e. generate related Cucumber .feature files), and run it in TeamCity against the code that actually implements each step that are part of those scenarios.

Finally, we can then submit the results back to JIRA and they'll be reflected on the related entities.

Overall, our Build Configuration is composed of 3 basic steps.

Administration / <Root project> / cucumber_xray_tests-local-git

Run ...

Build

General Settings

Version Control Settings 1

Build Steps 3

Triggers 1

Failure Conditions

Build Features

Dependencies

Parameters

Agent Requirements

Last edited 16 hours ago by admin (view history)

Build Steps

In this section you can configure the sequence of build steps to be executed. Each build step is represented by a build runner and provides integration with a

+ Add build step

Reorder build steps

Auto-detect build steps

Build Step	Parameters Description
1. export cucumber features	Command Line Custom script: curl -u admin:admin "http://192.168.56.1... (and 1 more line) Execute: If all previous steps finished successfully
2. run cucumber scenarios	Command Line Custom script: #!/bin/bash --login (and 4 more lines) Execute: If all previous steps finished successfully
3. Import results to Xray	Command Line Custom script: curl -v -H "Content-Type: application/js... Execute: Even if some of the previous steps failed

Exporting Cucumber features

We start by extracting the tests specification out of JIRA and generate the proper .feature files.

The export can take as input issue keys of requirements, Test Executions, Test Plans or a filter id, which will be the one we'll use.

For this, we'll invoke the REST API ([Exporting Cucumber Tests - REST](#)) in order to obtain a .zip file containing the .feature files. We'll be using a Build Step of type "Command Line" for this purpose, along with "curl" utility to ease making the HTTP request.

Build Step (1 of 3): export cucumber features | ▾

Runner type:	Command Line Simple command execution
Step name:	export cucumber features Optional, specify to distinguish this build step from other steps.
Execute step: ⓘ	If all previous steps finished successfully Specify the step execution policy.
Working directory: ⓘ	cucumber_xray_tests Optional, set if differs from the checkout directory.
Run:	Custom script
Custom script: *	Enter build script content: curl -u %jira_user%:%jira_password% "%jira_base_url% unzip -o features/features.zip -d features/

The complete script content of the "custom script" field above is:

```
curl -u %jira_user%:%jira_password% "%jira_base_url%/rest/raven/1.0/export/test?filter=11400&fz=true" -o  
features/features.zip  
unzip -o features/features.zip -d features/
```

Notice that we're unzipping the .feature files to a local directory, so we're able to run them.

Run Cucumber scenarios

The exact syntax for running the Cucumber scenarios depends on the Cucumber implementation being used; in this case we're using Ruby's variant. Therefore we're basically just invoking "cucumber" command with an option to generate a JSON report (e.g. "data.json").

Build Step (2 of 3): run cucumber scenarios | ▾

Runner type:	Command Line Simple command execution
Step name:	run cucumber scenarios Optional, specify to distinguish this build step from other steps.
Execute step: ⓘ	If all previous steps finished successfully Specify the step execution policy.
Working directory: ⓘ	cucumber_xray_tests Optional, set if differs from the checkout directory.
Run:	Custom script
Custom script: *	Enter build script content: #!/bin/bash --login rvm use 2.3 cucumber -x -f json -o data.json :

You may have noticed a trick in the cucumber line above, in the end of the command (i.e. "... || :"). That ensures that cucumber returns with exit code 0 (i.e. success), so the build may proceed.

Import execution results

In order to submit the results, we'll need to add a Build Step of type "Command Line", where we'll invoke the REST API, submitting the Cucumber JSON report generated in the previous step.

We also make sure this step is called always.

Build Step (3 of 3): Import results to Xray | ▾

Runner type:

Command Line ▾
Simple command execution

Step name:

Import results to Xray
Optional, specify to distinguish this build step from other steps.

Execute step: ⓘ

Even if some of the previous steps failed ▾
Specify the step execution policy.

Working directory: ⓘ

cucumber_xray_tests
Optional, set if differs from the checkout directory.

Run:

Custom script ▾

Custom script: *

Enter build script content:
curl -v -H "Content-Type: application/json" -X POST

The complete script content of the "custom script" field above is:

```
curl -v -H "Content-Type: application/json" -X POST -u %jira_user%:%jira_password% --data @data.json "%jira_base_url%/rest/raven/1.0/import/execution/cucumber"
```

You may notice that we're using some parameters related with the Jira server, that we've configured at project level.

ⓘ Please note

The user present in the configuration below must exist in the JIRA instance and have permission to Create Test and Test Execution Issues

Xray provides a free plugin for TeamCity. Please use it instead, as described in [Integration with TeamCity \(legacy\)](#).
Administration / 🗄 <Root project>

🗄 cucumber_xray_tests-local-git ⓘ 1 info item

General Settings

VCS Roots 1

Report Tabs 1

Parameters 3

Builds Schedule

Connections

SSH Keys

Meta-Runners

Maven Settings

+ Add new parameter

Configuration Parameters

Configuration parameters are not passed into build, can be used in references only. ⓘ

Name	Value
jira_base_url	http://192.168.56.102
jira_password	*****
jira_user	admin