Integration with Maven

Overview

Integration with Maven is possible using an open-source Maven plugin sponsored by the Xray team; the Maven plugin is a wrapper that invokes Xray API's to achieve its goals.

This plugin allows uploading test results to Xray, supporting a wide range of test reports/formats; it also allows the import and export of Cucumber/Gherkin scenarios to/from Xray, and thus allow the implementation of the related automation code.

More info can be found on the respective GitHub project. The previous proprietary Maven plugin provided by the Xray team, which only partially supported Xray server/DC, has been deprecated.

> **Please note**
>
> Support for the open-source plugin should be handled through the respective GitHub project, and is in line with regular open-source projects. In other words, users can report issues, ideas, but they are also encouraged to make contributions. There is no SLA whatsoever for issues raised on this open-source project, as issues will be handled on a best effort by the community itself.
>
> If you're using the previous proprietary Maven plugin, you still have access to official support but please note that it has been deprecated and it is highly recommended to this one instead.

Installation and Configuration

This plugin is available on (Maven) Central Repository, which is configured by default in your Maven instalation.

Add the following dependency to your `pom.xml`, where the `<configuration>` is optional and should be adapted to your use case.

Please note

Plese make sure whether you're using Xray on Jira Cloud or Xray on Jira server/datacenter, as these are two similar but different products with slightly differente capabilites.

pom.xml sample

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <projectKey>CALC</projectKey>
                <fixVersion>1.0</fixVersion>
                <reportFormat>junit</reportFormat>
                <reportFile>target/junit.xml</reportFile>
            </configuration>
        </dependency>
```

Configuration parameters (e.g., "reportFormat") can also be specified from the command line using `-D` (e.g., "-Dxray.reportFormat=junit"). In this case, the parameters have the `xray.` prefix.

Configuration made directly on the pom.xml file has higher priority over command line arguments.

There are a set of common configurations related to the Xray details and its authentication. Besides, each task has its own configuration parameters as shown ahead.

Common configurations

The base configurations depend on whether you're using Xray on Jira Cloud or on Jira server/datacenter.

Xray server/datacenter users (i.e., using Xray on Jira server/datacenter)

Xray on Jira server/datacenter is built on top of Jira's REST API, an thus reuses the built-in capabilities provided by Jira itself.

Auhentication is done either using a Jira user's username and password, or eventually a Jira Personal Access Token; none of these are managed directly by Xray but by Jira instead.

parameter	command line parameter	description	example
`cloud`	`xray.cloud`	set to false if not using Xray cloud (default: false)	false
`jiraBaseUrl`	`xray.jiraBaseUrl`	Jira server/DC base URL	http://10.0.0.1/
`jiraUsername`	`xray.jiraUsername`	username of Jira user to use on API calls	someuser
`jiraPassword`	`xray.jiraPassword`	password of Jira user to use on API calls	somepass
`jiraToken`	`xray.jiraToken`	Jira PAT (Personal Access Token) used instead of username/password	
`ignoreSslErrors`	`xray.ignoreSslErrors`	ignore SSL errors, e.g., expired certificate (default: false)	
`timeout`	`xray.timeout`	connection timeout in seconds (default: 50)	

Xray cloud users (i.e., using Xray on Jira cloud)

Xray on Jira cloud uses its own mechanism (i.e. API key pairs) for authenticating requests; in order to obtain the API key (client id + client secret pair) please ask you Jira admin.

parameter	command line parameter	description	example
`cloud`	`xray.cloud`	set to true if using Xray cloud (default: false)	true
`clientId`	`xray.clientId`	client id of the API key configured on Xray Cloud	xxxx...
`clientSecret`	`xray.clientSecret`	client id of the API key configured on Xray Cloud	xxxx...
`timeout`	`xray.timeout`	connection timeout in seconds (default: 50)	50

How to use

This plugin provides these tasks:

- `xray:import-results`
- `xray:import-features`
- `xray:export-features`

Importing test automation results

In order to import test results, we need to use the `xray:import-results` task.

```
mvn clean compile test xray:import-results
```

The `pom.xml` needs to be configured properly (see available configurations). As an alternative to hardcode the configurations, it's also possible to pass them right from the command line as mentioned earlier, or even have some on the pom.xml and another specificed through command line parameters.

```
mvn clean compile test xray:import-results -Dxray.reportFormat=junit -Dxray.reportFile=results/junit.xml
```

Xray server/DC and Xray cloud support mostly the same formats; please check the respective product documentation as restrictions may apply.

Configuration options

There are two ways of importing results. We can either choose one or the other but not both.

The first one is more simple (recommended for most uses) and also known as "standard" (due to how it is called in terms of REST API) where we provide all or a mix of predefined and common parameters (e.g., projectKey, version) that are enough for most usage scenarios.

setting	command line parameter	description	mandatory/optional	example
`reportFormat`	`xray.reportFormat`	format of the report (junit, testng, nunit, xunit, robot, xunit, cucumber, behave)	mandatory	junit
`reportFile`	`xray.reportFile`	file with the test results (relative or absolute path); it can also be a directory (all .xml files will be imported in this case); finally, it can also be a regex that applies to the current working directory	mandatory	target/junit.xml
`projectKey`	`xray.projectKey`	key of Jira project where to import the results	mandatory (doesn't apply to "cucumber" or "behave" report formats, for legacy reasons)	CALC
`testExecKey`	`xray.testExecKey`	issue key of Test Execution, in case we want to update the results on it	optional	CALC-2
`testPlanKey`	`xray.testPlanKey`	issue key of Test Plan to link the results to	optional	CALC-1
`version`	`xray.version`	version of the SUT, that corresponds to the Jira project version/release; it will be assigned to the Test Execution issue using the "fixVersion(s)" field	optional	1.0
`revision`	`xray.revision`	source code revision or a build identifier	optional	123
`testEnvironment`	`xray.testEnvironment`	usually, a test environment name/identifier (e.g., browser vendor, OS versio , mobile device, testing stage); multiple test environments may be specified though using ";" as delimiter	optional	chrome
`testInfoJson`	`xray.testInfoJson`	path to a JSON file containing attributes to apply on the Test issues that may be created, following Jira issue update syntax	optional	-
`testExecInfoJson`	`xray.testExecInfoJson`	path to a JSON file containing attributes to apply on the Test Execution issue that may be created, following Jira issue update syntax	optional	-

`abortOnError`	abort, if multiple results are being imported, and exit with error if uploading results fails	optional	-	

There is another way importing results though, that allow us to customize any field on the Test Execution issue or even on the Test issues that may be created; in this case, we need to pass the `testExecInfoJson` and/or the `testInfoJson` fields. This approach, also known as "multipart" due to the Xray REST API endpoint it uses, even though more flexible will require us to specify common fields such as the project key, version, and other, within the respective JSON field.

setting	command line parameter	description	mandatory /optional	example
`reportFormat`	`xray.reportFormat`	format of the report (junit, testng, nunit, xunit, robot, xunit, cucumber, behave)	mandatory	junit
`reportFile`	`xray.reportFile`	file with the test results (relative or absolute path); it can also be a directory (all .xml files will be imported in this case); finally, it can also be a regex that applies to the current working directory	mandatory	target/junit.xml
`testInfoJson`	`xray.testInfoJson`	path to a JSON file containing attributes to apply on the Test issues that may be created, following Jira issue update syntax	optional	testInfo.json
`testExecInfoJson`	`xray.testExecInfoJson`	path to a JSON file containing attributes to apply on the Test Execution issue that may be created, following Jira issue update syntax	optional	testExecInfo.json
`abortOnError`	abort, if multiple results are being imported, and exit with error if uploading results fails. "false" by default.	optional	-	true

Examples

Import a file with Junit XML test results to Xray; assign them to a project, version, and Test Plan

In this example we will:

- upload a JUnit XML report to Xray
- create a Test Execution and assign to a given Jira project, release (i.e. project's FixVersion), and Test Plan

Xray Cloud: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <projectKey>CALC</projectKey>
                <fixVersion>1.0</fixVersion>
                <testPlanKey>CALC-1200</testPlanKey>
                <reportFormat>junit</reportFormat>
                <reportFile>target/junit.xml</reportFile>
            </configuration>
        </dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <projectKey>CALC</projectKey>
                <fixVersion>1.0</fixVersion>
                <testPlanKey>CALC-1200</testPlanKey>
                <reportFormat>junit</reportFormat>
                <reportFile>target/junit.xml</reportFile>
                                <abortOnError>true></abortOnError>
            </configuration>
        </dependency>
```

To import the test results, we need to invoke the respective task.

```
mvn clean compile test xray:import-results
```

Import multiple Junit XML test results to Xray; assign them to a project, version, and Test Plan

In this example we will:

- upload mutiple JUnit XML reports to Xray, based on a given regular expression
- create a Test Execution for each imported file, and assign to a given Jira project, release (i.e. project's FixVersion), and Test Plan
- abort the process, if any of the import fails

Xray Cloud: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <projectKey>CALC</projectKey>
                <fixVersion>1.0</fixVersion>
                <testPlanKey>CALC-1200</testPlanKey>
                <reportFormat>junit</reportFormat>
                <reportFile>target/*.xml</reportFile>
                                <abortOnError>true></abortOnError>
            </configuration>
        </dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <projectKey>CALC</projectKey>
                <fixVersion>1.0</fixVersion>
                <testPlanKey>CALC-1200</testPlanKey>
                <reportFormat>junit</reportFormat>
                <reportFile>target/*.xml</reportFile>
                                <abortOnError>true></abortOnError>
            </configuration>
        </dependency>
```

To import the test results, we need to invoke the respective task.

```
mvn clean compile test xray:import-results
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:import-features -Dxray.reportFile=target/*.xml
```

Import a file with Junit XML test results to Xray, and customize fields on the corresponding Test Execution issue

In this example we will:

- upload a JUnit XML report to Xray
- create a Test Execution and assign to a given Jira project, release (i.e. project's FixVersion)
- additionaly, customize some fields on the Test Execution issue (e.g., set a label, description, a custom field by its id obtainable from Jira administration)

Xray Cloud: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <reportFormat>junit</reportFormat>
                <reportFile>target/junit.xml</reportFile>
                                <testExecInfoJson>testExecInfo.json</testExecInfoJson>
                                <testInfoJson>testInfo.json</testInfoJson>
                                <abortOnError>true></abortOnError>
            </configuration>
        </dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <reportFormat>junit</reportFormat>
                <reportFile>target/junit.xml</reportFile>
                                <testExecInfoJson>testExecInfo.json</testExecInfoJson>
                                <testInfoJson>testInfo.json</testInfoJson>
                                <abortOnError>true></abortOnError>
            </configuration>
        </dependency>
```

The `testExecInfo.json` contents are as follows.

testExecInfo.json sample

```
{
    "fields": {
        "project": {
            "key": "CALC"
        },
        "summary": "Results for some automated tests",
        "description": "For more info please check [here|https://www.example.com]",
        "issuetype": {
            "name": "Test Execution"
        },
        "customfield_11805" : [
            "iOS"
        ],
        "fixVersions" :
                [
                 {
                        "name": "1.0"
                 }
                ]
    }
}
```

To import the test results, we need to invoke the respective task.

```
mvn clean compile test xray:import-results
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:import-features -Dxray.reportFile=target/*.xml -Dxray.testExecInfoJson=testExecInfo.json
```

Import a file with Junit XML test results to Xray, and customize fields on corresponding Test issues

In this example we will:

- upload a JUnit XML report to Xray
- create a Test Execution and assign to a given Jira project, release (i.e. project's FixVersion)
- customize some fields on the Test issues (e.g., set a label) that will be provisioned the first time results are imported (if the Tests don't already exist)

Xray Cloud: pom.xml snippet

```
<dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.3</version>
    <scope>test</scope>
    <configuration>
        <cloud>true</cloud>
        <clientId>215FFD69FE46447280000000000</clientId>
        <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
        <reportFormat>junit</reportFormat>
        <reportFile>target/junit.xml</reportFile>
                        <testExecInfoJson>testExecInfo.json</testExecInfoJson>
                        <testInfoJson>testInfo.json</testInfoJson>
                        <abortOnError>true></abortOnError>
    </configuration>
</dependency>
```

Xray server/datacenter: pom.xml snippet

```
<dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.3</version>
    <scope>test</scope>
    <configuration>
        <cloud>false</cloud>
                        <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
        <jiraUsername>someuser</jiraUsername>
        <jiraPassword>somepass</jiraPassword>
        <reportFormat>junit</reportFormat>
        <reportFile>target/junit.xml</reportFile>
                        <testExecInfoJson>testExecInfo.json</testExecInfoJson>
                        <testInfoJson>testInfo.json</testInfoJson>
                        <abortOnError>true></abortOnError>
    </configuration>
</dependency>
```

The `testExecInfo.json` contents are as follows.

testExecInfo.json sample

```
{
    "fields": {
        "project": {
            "key": "CALC"
        },
        "summary": "Test Execution for some automated tests",
        "issuetype": {
            "name": "Test Execution"
        },
        "fixVersions" :
                [
                 {
                        "name": "1.0"
                 }
                ]
    }
}
```

The `testInfo.json` contents are as follows.

testInfo.json sample

```
{
    "fields": {
        "description": "Automated Test",
        "labels": [
            "Automation"
        ]
    }
}
```

To import the test results, we need to invoke the respective task.

```
mvn clean compile test xray:import-results
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:import-features -Dxray.reportFile=target/*.xml -Dxray.testExecInfoJson=testExecInfo.json -Dxray.
testInfoJson=testInfo.json
```

Importing/synchronizing Cucumber .feature files to Xray

One of the possible workflows for using Gherkin-based frameworks is to use git (or other versioning control system) as the master to store the corresponding .feature files (more info here). In order to provide visibility of test results for these tests (i.e. gherkin Scenarios), these need to exist in Xray beforehand. Therefore, we need to import/synchronize them to Xray. Note that there is no direct integration; the integration is adhoc, i.e., the following task is run on a local copy of the respository where the .features are stored in.

```
mvn clean compile test xray:import-features -Dxray.inputFeatures=features/
```

Note: how Xray relates the Scenarios/Background to the corresponding Test or Precondition issues is described in Xray technical documentation (e.g., Xray cloud docs, Xray server/DC docs).

Configuration options

parameter	command line parameter	description	mandatory /optional	example
`projectKey`	`xray.projectKey`	key of Jira project where to import the Cucumber Scenarios/Backgrounds as Test and Precondition issues	mandatory (if projectId not used)	CALC
`projectId`	`xray.projectId`	**Xray cloud only**: id of Jira project where to import the Cucumber Scenarios/Backgrounds as Test and Precondition issues	mandatory (if projectKey not used)	1000
`source`	`xray.source`		optional; only applies to Xray cloud	
`testInfoJson`	`xray.testInfoJson`	path to a JSON file containing attributes to apply on the Test issues that may be created, following Jira issue update syntax	optional	-
`precondInfoJson`	`xray.precondInfoJson`	path to a JSON file containing attributes to apply on the Precondition issues that may be created, following Jira issue update syntax	optional	-
`inputFeatures`	`xray.inputFeatures`	either a .feature file, a directory containing .feature files, or a zipped file containing .feature files	mandatory	features/
`updateRepository`	`xray.updateRepository`	create folder structure in Test Repository based on the folder structure in the .zip file containing the .feature files; default is false. Only supported on Xray server/DC.	optional	true

Examples

Importing Scenarios and Backgrounds from existing .feature files in a folder to Xray

In this example we will:

- process all .feature files in the given folder
- create or update existing Cucumber/Gherkin Test(s) and/or Precondition(s) issues in Xray

Xray Cloud: pom.xml snippet

```
<dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.3</version>
    <scope>test</scope>
    <configuration>
        <cloud>true</cloud>
        <clientId>215FFD69FE46447280000000000</clientId>
        <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
        <featuresPath>features/</featuresPath>
    </configuration>
</dependency>
```

Xray server/datacenter: pom.xml snippet

```
<dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.3</version>
    <scope>test</scope>
    <configuration>
        <cloud>false</cloud>
                        <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
        <jiraUsername>someuser</jiraUsername>
        <jiraPassword>somepass</jiraPassword>
        <featuresPath>features/</featuresPath>
    </configuration>
</dependency>
```

To import the Gherkin scenarios, we need to invoke the respective task.

```
mvn xray:import-features
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:import-features -Dxray.featuresPath=features/
```

Importing Scenarios and Backgrounds from existing .feature files in a zipped file to Xray

In this example we will:

- process all .feature files in the given .zip file
- create or update existing Cucumber/Gherkin Test(s) and/or Precondition(s) issues in Xray

This can be done just once to import the test scenarios to Xray, or can be done multiple times, to "synchronize" (i.e., update) the scenarios in Xray.

Xray Cloud: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <featuresPath>gherkin_features.zip</featuresPath>
            </configuration>
        </dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <featuresPath>gherkin_features.zip</featuresPath>
            </configuration>
        </dependency>
```

To import the Gherkin scenarios, we need to invoke the respective task.

```
mvn xray:import-features
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:import-features -Dxray.featuresPath=gherkin_features.zip
```

Exporting/generating Cucumber .feature files from Xray

No matter which worflow you decide to use for dealing with Gherkin-based tests (e.g., Cucumber .feature files and corresponding Scenarios), as part of that workflow comes the need to extract/generate the .feature files based on Scenarios or Backgrounds detailed in Xray using Test or Precondition issues. This plugin provides a task for this purpose. It will download/generate .feature files to a local folder from existing information in Xray. The Feature, Scenario elements will be properly tagged with info from Xray.

```
mvn xray:export-features -Dxray.issueKeys=CALC-1,CALC-2 -Dxray.outputDir=features/
```

Files on the destination folder will be overwritten; however, if this directory contains other information (including other .feature files) you may need to remove them before generating the .feature files into this directory.

Note: how Xray generates the .feature files with the Scenarios/Background from existing Test or Precondition issues is described in Xray technical documentation (e.g., Xray cloud docs, Xray server/DC docs).

Configurations for exporting/generating Cucumber .feature files from Xray

parameter	command line parameter	description	mandatory/optional	example
`issueKeys`	`xray.issueKeys`	issue keys of direct or indirect references to Cucumber/Gherkin tests/scenarios (e.g., Test issue keys), delimited by comma	mandatory (or optional if filterId is used instead)	CALC-1, CALC-2
`filterId`	`xray.filterId`	id of the Jira filter containing direct or indirect references to Cucumber/Gherkin tests/scenarios	mandatory (or optional if issueKeys is used instead)	12000
`outputDir`	`xray.outputDir`	output directory where the .feature files should be extracted to	mandatory	features/

Examples

Export Cucumber/Gherkin test scenarios from Xray to a local directory, based on the given issue keys

In this example we will:

- export two existing Cucumber/Gherkin Test issues from Xray; these issue keys could also be referring to story issues, Test Plans, etc (Xray will find out the related Cucumber/Gherkin Tests and export those)
- generate the corresponding .feature file(s), on a local directory

Xray Cloud: pom.xml snippet

```
<dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.3</version>
    <scope>test</scope>
    <configuration>
        <cloud>true</cloud>
        <clientId>215FFD69FE46447280000000000</clientId>
        <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
        <issueKeys>CALC-1,CALC-2</issueKeys>
        <outputDir>features/</outputDir>
    </configuration>
</dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <issueKeys>CALC-1,CALC-2</issueKeys>
                <outputDir>features/</outputDir>
            </configuration>
        </dependency>
```

To export the Cucumber/Gherkin tests and generate the corresponding .feature file(s), we need to invoke the respective task.

```
mvn xray:export-features
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:export-features -Dxray.issueKeys=CALC-1,CALC-2 -Dxray.outputDir=features/
```

Export Cucumber/Gherkin test scenarios from Xray to a local directory, based on the given Jira filter id

In this example we will:

- export two existing Cucumber/Gherkin Test issues from Xray, based on a given Jira filter id; the filter can include Tests directly or related issues (e.g., story issues, Test Plans, etc), and Xray will then find out the related Cucumber/Gherkin Tests and export those
- generate the corresponding .feature file(s), on a local directory

Xray Cloud: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>true</cloud>
                <clientId>215FFD69FE46447280000000000</clientId>
                <clientSecret>1c00f8f22f56a8684d7c18cd61470000000</clientSecret>
                <filterId>12345</filterId>
                <outputDir>features/</outputDir>
            </configuration>
        </dependency>
```

Xray server/datacenter: pom.xml snippet

```
        <dependency>
            <groupId>app.getxray</groupId>
            <artifactId>xray-maven-plugin</artifactId>
            <version>0.7.3</version>
            <scope>test</scope>
            <configuration>
                <cloud>false</cloud>
                                <jiraBaseUrl>https://myjiraserver</jiraBaseUrl>
                <jiraUsername>someuser</jiraUsername>
                <jiraPassword>somepass</jiraPassword>
                <filterId>12345</filterId>
                <outputDir>features/</outputDir>
            </configuration>
        </dependency>
```

To export the Cucumber/Gherkin tests and generate the corresponding .feature file(s), we need to invoke the respective task.

```
mvn xray:export-features
```

We could also specify some of these options from the command line instead of having them hardcoded in the `pom.xml` file.

```
mvn xray:export-features -Dxray.filterId=12345 -Dxray.outputDir=features/
```

FAQ

- Is this the same maven plugin as the original one made by the Xray team?
  - No. This is a totally new one, made from scratch and open-source.
- If we have questions/support issues, where should those be addressed?
  - It's an open-source project, so it should be handled in the GitHub project and supported by the community. If you want to use the previous, proprietary plugin, you can do so and that has commercial support, if you have a valid license; please note that the proprietary Maven plugin has been deprecated meanwhile.
- Are the underlying APIs the same for Xray server/datacenter and Xray Cloud? Are the available options the same? Are the supported test automation report formats the same?
  - Not exactly. Xray server/datacenter and Xray cloud, even though similar, are actually distinct products; besides Jira server/datacenter and Jira cloud are different between themselves and have different capabilities. This plugin makes use of the available REST APIs for Xray server/datancer and Xray cloud, so you should check them to see exactly what is supported for your environment.