OpenL Tablets Maven Plugin Guide

OpenL Tablets 5.12

OpenL Tablets BRMS

Table of Contents

1. **Table of Contents** i
2. **Introduction** 1
3. **Goals** .....
3.1. Generate goal 3
3.2. Compile goal 8
3.3. Test goal 9
3.4. Help goal 11
4. **Usage** 13
5. **Examples** .....
5.1. Configuration with all goals 18
5.2. Generate a project with working example 19

1 Introduction

1.1 OpenL Maven Plugin

Access to rules and data in Excel tables is realized through OpenL Tablets API. OpenL Tablets provides wrappers to developers to facilitate easier usage.

This plugin can generate interface that can be used to access the rules. Also with this plugin you can validate rules during compilation phase and run OpenL Tablets tests.

1.1.1 Goals Overview

General information about the goals:

- openl:generate Generate OpenL interface, domain classes and project descriptor.
- openl:compile Compile OpenL project.
- openl:test Run OpenL tests.
- openl:help Display help information on openl-maven-plugin.

1.1.2 Usage

General instructions on how to use the Plugin Name can be found on the usage page. Some more specific use cases are described in the examples given below.

You should specify the version in your project's plugin configuration:

```
<project>
  ...
  <build>
    <!-- To define the plugin version in your parent POM -->
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.version}</version>
        </plugin>
        ...
      </plugins>
    </pluginManagement>
    <!-- To use the plugin goals in your POM or parent POM -->
    <plugins>
      <plugin>
        <groupId>org.openl.rules</groupId>
        <artifactId>openl-maven-plugin</artifactId>
        <version>${openl.version}</version>
      </plugin>
      ...
    </plugins>
  </build>
  ...
</project>
```

1.1.3 Examples

To provide you with better understanding of some usages of the Plugin Name, you can take a look into the following examples:

- Configuration with all goals
- Generate a project with working example

2 Generate goal

2.1 openl:generate

Full name:

org.openl.rules:openl-maven-plugin:5.12.0:generate

Description:

Generate OpenL interface, domain classes, project descriptor and unit tests

Attributes:

- Requires a Maven project to be executed.
- Binds by default to the lifecycle phase: `generate-sources`.

2.1.1 Required Parameters

Name	Type	Description
generateInterfaces	`JavaAntTask[]`	Tasks that will generate classes. Object Properties

Name	Type	Requ	Description
srcFile	String	true	Reference to the Excel file for which an interface class must be generated.
targetClass	String	true	Full name of the interface class to be generated. OpenL Tablets WebStudio recognizes modules in projects by interface classes and uses their names in the user interface. If there are multiple wrappers with identical names, only one of them is recognized as a module in OpenL Tablets WebStudio.
displayName	String	false	End user oriented title of the file that appears in OpenL Tablets WebStudio. Default value is Excel file name without extension.
targetSrcDir	String	false	Folder where the generated interface class must be placed. For example: "src/main/java". Default value is: "${project.build.sourceDirectory}"
openlName	String	false	OpenL configuration to be used. For OpenL Tablets, the following value must always be used: org.openl.xls. Default value is: "org.openl.xls"

userHome	String	false	Location of user-defined resources relative to the current OpenL Tablets project. Default value is: "."
userClassPath	String	false	Reference to the folder with additional compiled classes that will be imported by the module when the interface is generated. Default value is: null.
ignoreTestMeth	boolean	false	If true - test methods will not be added to interface class. Used only in JavaInterfaceAntTask. Default value is: true.
generateUnitTe	boolean	false	Overwrites base `generateUnitTests` value
unitTestTempla	String	false	Overwrites base `unitTestTemplatePath` value
overwriteUnitTe	boolean	false	Overwrites base `overwriteUnitTests` value

2.1.2 Optional Parameters

Name	Type	Description
classpaths	`String[]`	Default classpath entries in rules.xml. Default value is {"."} Used only if createProjectDescriptor == true.
createProjectDescriptor	`boolean`	If true - rules.xml will be generated if it doesn't exist. If false, rules.xml will not be generated. Default value is "true". **Default value is**: `true`.
generateUnitTests	`Boolean`	If true - JUnit tests for OpenL Tablets Test tables will be generated. Default value is "false" **Default value is**: `false`.
openlOutputDirectory	`String`	Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl". **Default value is**: `${project.build.directory}/openl`.
openlResourcesDirectory	`String`	Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl". **Default value is**: `${project.basedir}/src/main/openl`.
overwriteProjectDescriptor	`boolean`	If true - rules.xml will be overwritten on each run. If false, rules.xml generation will be skipped if it exists. Makes sense only if `createProjectDescriptor` == true. Default value is "true".

		Default value is: `true`.
overwriteUnitTests	`Boolean`	If true - existing JUnit tests will be overwritten. If false - only absent tests will be generated, others will be skipped. **Default value is**: `false`.
projectName	`String`	Default project name in rules.xml. If omitted - the name of a first module in the project is used. Used only if createProjectDescriptor == true.
unitTestTemplatePath	`String`	Path to Velocity template for generated unit tests. If omitted - default template will be used. Available in template variables:

Name	Description
openlInterfacePackage	Package of generated interface class
openlInterfaceClass	Generated interface class name
testMethodNames	Available test method names
projectRoot	Root directory of OpenL project
srcFile	Reference to the Excel file for which an interface class must be generated.
StringUtils	Apache commons utility class

Default value is: `org/openl/rules/maven/JUnitTestTemplate.vm`.

2.1.3 Parameter Details

classpaths:

Default classpath entries in rules.xml. Default value is {"."} Used only if createProjectDescriptor == true.

- **Type**: `java.lang.String[]`
- **Required**: `No`

---

createProjectDescriptor:

If true - rules.xml will be generated if it doesn't exist. If false, rules.xml will not be generated. Default value is "true".

- **Type**: `boolean`
- **Required**: `No`
- **Default**: `true`

---

generateInterfaces:

Tasks that will generate classes.

Object Properties

Name	Type	Required	Description
srcFile	String	true	Reference to the Excel file for which an interface class must be generated.
targetClass	String	true	Full name of the interface class to be generated. OpenL Tablets WebStudio recognizes modules in projects by interface classes and uses their names in the user interface. If there are multiple wrappers with identical names, only one of them is recognized as a module in OpenL Tablets WebStudio.
displayName	String	false	End user oriented title of the file that appears in OpenL Tablets WebStudio. Default value is Excel file name without extension.
targetSrcDir	String	false	Folder where the generated interface class must be placed. For example: "src/main/java". Default value is: "${project.build.sourceDirectory}"
openlName	String	false	OpenL configuration to be used. For OpenL Tablets, the following value must always be used: org.openl.xls. Default value is: "org.openl.xls"
userHome	String	false	Location of user-defined resources relative to the current OpenL Tablets project. Default value is: "."
userClassPath	String	false	Reference to the folder with additional compiled classes that will be imported by the module when the interface is generated. Default value is: null.
ignoreTestMethods	boolean	false	If true - test methods will not be added to interface class. Used only in JavaInterfaceAntTask. Default value is: true.
generateUnitTests	boolean	false	Overwrites base `generateUnitTests` value
unitTestTemplatePath	String	false	Overwrites base `unitTestTemplatePath` value
overwriteUnitTests	boolean	false	Overwrites base `overwriteUnitTests` value

- **Type**: `org.openl.conf.ant.JavaAntTask[]`
- **Required**: `Yes`

---

generateUnitTests:

If true - JUnit tests for OpenL Tablets Test tables will be generated. Default value is "false"

- **Type**: `java.lang.Boolean`
- **Required**: `No`
- **Default**: `false`

---

openlOutputDirectory:

Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.build.directory}/openl`

---

openlResourcesDirectory:

Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.basedir}/src/main/openl`

---

overwriteProjectDescriptor:

If true - rules.xml will be overwritten on each run. If false, rules.xml generation will be skipped if it exists. Makes sense only if `createProjectDescriptor` == true. Default value is "true".

- **Type**: `boolean`
- **Required**: `No`
- **Default**: `true`

---

overwriteUnitTests:

If true - existing JUnit tests will be overwritten. If false - only absent tests will be generated, others will be skipped.

- **Type**: `java.lang.Boolean`
- **Required**: `No`
- **Default**: `false`

---

projectName:

Default project name in rules.xml. If omitted - the name of a first module in the project is used. Used only if createProjectDescriptor == true.

- **Type**: `java.lang.String`
- **Required**: `No`

---

unitTestTemplatePath:

Path to Velocity template for generated unit tests. If omitted - default template will be used. Available in template variables:

Name	Description
openlInterfacePackage	Package of generated interface class
openlInterfaceClass	Generated interface class name
testMethodNames	Available test method names
projectRoot	Root directory of OpenL project
srcFile	Reference to the Excel file for which an interface class must be generated.
StringUtils	Apache commons utility class

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `org/openl/rules/maven/JUnitTestTemplate.vm`

3 Compile goal

3.1 openl:compile

Full name:

org.openl.rules:openl-maven-plugin:5.12.0:compile

Description:

Compile and validate OpenL project

Attributes:

- Requires a Maven project to be executed.
- Binds by default to the lifecycle phase: `compile`.

3.1.1 Optional Parameters

Name	Type	Description
openlOutputDirectory	`String`	Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl". **Default value is**: `${project.build.directory}/openl`.
openlResourcesDirectory	`String`	Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl". **Default value is**: `${project.basedir}/src/main/openl`.

3.1.2 Parameter Details

openlOutputDirectory:

Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.build.directory}/openl`

---

openlResourcesDirectory:

Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.basedir}/src/main/openl`

4 Test goal

4.1 openl:test

Full name:

org.openl.rules:openl-maven-plugin:5.12.0:test

Description:

Run OpenL tests

Attributes:

- Requires a Maven project to be executed.
- Binds by default to the lifecycle phase: `test`.

4.1.1 Optional Parameters

Name	Type	Description
openlOutputDirectory	`String`	Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl". **Default value is**: `${project.build.directory}/openl`.
openlResourcesDirectory	`String`	Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl". **Default value is**: `${project.basedir}/src/main/openl`.
skipTests	`boolean`	Set this to 'true' to skip running OpenL tests. **User property is**: `skipTests`.

4.1.2 Parameter Details

openlOutputDirectory:

Folder used by OpenL to compile rules. For example: ${project.build.directory}/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.build.directory}/openl`

---

openlResourcesDirectory:

Folder that contains all OpenL-related resources (OpenL rules, project descriptor etc.). For example: "${project.basedir}/src/main/openl".

- **Type**: `java.lang.String`
- **Required**: `No`
- **Default**: `${project.basedir}/src/main/openl`

---

skipTests:

Set this to 'true' to skip running OpenL tests.

- **Type**: `boolean`
- **Required**: `No`
- **User Property**: `skipTests`

5 Help goal

5.1 openl:help

Full name:

org.openl.rules:openl-maven-plugin:5.12.0:help

Description:

Display help information on openl-maven-plugin.
Call `mvn openl:help -Ddetail=true -Dgoal=<goal-name>` to display parameter details.

Attributes:

- The goal is thread-safe and supports parallel builds.

5.1.1 Optional Parameters

Name	Type	Description
detail	`boolean`	If `true`, display all settable properties for each goal. **Default value is**: `false`. **User property is**: `detail`.
goal	`String`	The name of the goal for which to show help. If unspecified, all goals will be displayed. **User property is**: `goal`.
indentSize	`int`	The number of spaces per indentation level, should be positive. **Default value is**: 2. **User property is**: `indentSize`.
lineLength	`int`	The maximum length of a display line, should be positive. **Default value is**: `80`. **User property is**: `lineLength`.

5.1.2 Parameter Details

detail:

If `true`, display all settable properties for each goal.

- **Type**: `boolean`
- **Required**: `No`
- **User Property**: `detail`
- **Default**: `false`

---

goal:

The name of the goal for which to show help. If unspecified, all goals will be displayed.

- **Type**: `java.lang.String`
- **Required**: `No`
- **User Property**: `goal`

---

indentSize:

The number of spaces per indentation level, should be positive.

- **Type**: `int`
- **Required**: `No`
- **User Property**: `indentSize`
- **Default**: `2`

---

lineLength:

The maximum length of a display line, should be positive.

- **Type**: `int`
- **Required**: `No`
- **User Property**: `lineLength`
- **Default**: `80`

6 Usage

6.1 Usage

6.1.1 Directory structure

Take into account that OpenL Maven Plugin expects the following directory structure:

```
|- your-project/                    Project root folder
| |- pom.xml                        Maven project file
| |
| |- src/
| |
| | |- main/
| | |
| | | |- java/                      Contains java sources
| | | |
| | | |- resources/                 Contains java resources
| | | |
| | | |- openl/                     Contains all OpenL-related resources (rules, xml etc.)
| | | |
| | | |
| | | | |- rules.xml                Openl project descriptor (for OpenL only)
| | | | |- rules/
| | | | | |- TemplateRules.xls      File with rules
```

Note that all OpenL-related resources are located in `src/main/openl` directory. But you can change it to fit your needs (you should change `openlResourcesDirectory` parameter in maven plugin configuration).

Note: It's not recommended to put OpenL-related resources to the `src/main/resources` folder: in this case your OpenL resources will be inside jar file alongside with compiled java classes - most probably it's not what you wanted to do in production.

6.1.2 Configure interface, domain classes and project descriptor generation

The simplest way to generate interface for rules defined in the file TemplateRules.xls:

```
<build>
  [...]
  <plugins>
      [...]
      <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.rules.version}</version>
          <configuration>
              <generateInterfaces>
                  <generateInterface>
                      <srcFile>src/main/openl/rules/TemplateRules.xls</srcFile>
                      <targetClass>org.company.gen.TemplateRulesInterface</targetClass>
                  </generateInterface>
              </generateInterfaces>
          </configuration>
          <executions>
              <execution>
                  <goals>
                      <goal>generate</goal>
                  </goals>
              </execution>
          </executions>
      </plugin>

  </plugins>
  [...]
</build>
```

In this case classes and rules.xml will be generated on each maven run on generate-sources phase.

If you want to invoke class generation manually, you should remove `executions` node and then run in console when you need:

```
mvn openl:generate
```

More configuration options you can find on openl:generate goal page.

6.1.3 Configure OpenL project compilation and validation

```
<build>
  [...]
  <plugins>
      [...]
      <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.rules.version}</version>
          <executions>
              <execution>
                  <goals>
                      <goal>compile</goal>
                  </goals>
              </execution>
          </executions>
      </plugin>

  </plugins>
  [...]
</build>
```

More configuration options you can find on openl:compile goal page.

6.1.4 Configure OpenL project testing

The simplest way to invoke OpenL Tablets Test:

```
<build>
  [...]
  <plugins>
      [...]
      <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.rules.version}</version>
          <executions>
              <execution>
                  <goals>
                      <goal>test</goal>
                  </goals>
              </execution>
          </executions>
      </plugin>

  </plugins>
  [...]
</build>
```

More configuration options you can find on openl:test goal page.

But if you want to have more control on tests and want to invoke and debug them from java code, you can instead generate JUnit tests for them (in this case don't use the `test` goal). Configure the `generate` goal:

```
<build>
  [...]
  <plugins>
      [...]
      <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.rules.version}</version>
          <configuration>
              <generateUnitTests>true</generateUnitTests>
              <generateInterfaces>
                [...]
              </generateInterfaces>
          </configuration>
          <executions>
              <execution>
                  <goals>
                      <goal>generate</goal>
                  </goals>
              </execution>
          </executions>
      </plugin>

  </plugins>
  [...]
</build>
```

If you want to define your own template for JUnit tests, you should set `unitTestTemplatePath` parameter with a path to your Velocity template (for example: `<unitTestTemplatePath>src/test/resources/MyTemplate.vm</unitTestTemplatePath>`). Here is an example of such template:

```
#if ($openlInterfacePackage)
package $openlInterfacePackage;
#end

import org.junit.Before;
import org.junit.Test;
import org.openl.rules.runtime.RulesEngineFactory;
import org.openl.rules.testmethod.TestUnitsResults;

import java.io.File;

import static org.junit.Assert.assertTrue;

#set( $openlInterfaceClassWithTests = "${openlInterfaceClass}WithTests" )

public class ${openlInterfaceClass}Test {
    private static interface $openlInterfaceClassWithTests extends $openlInterfaceClass {
#foreach( $testMethodName in $testMethodNames )
        TestUnitsResults $testMethodName();
#end
    }

    private $openlInterfaceClassWithTests instance;

    @Before
    public void setUp() throws Exception {
        File xlsFile = new File("$projectRoot", "$srcFile");
        instance = new RulesEngineFactory<$openlInterfaceClassWithTests>(
            xlsFile,
            ${openlInterfaceClassWithTests}.class
        ).newEngineInstance();
    }

#foreach( $testMethodName in $testMethodNames )
    @Test
    public void test$StringUtils.capitalize($testMethodName)() throws Exception {
        TestUnitsResults results = instance.$testMethodName();
        assertTrue(results.toString(), results.getNumberOfFailures() == 0);
    }
#end
}
```

7 Configuration with all goals

7.1 Configuration with all OpenL Maven Plugin goals

You can fully configure rules.xml generation: set it's project id, project name, add classpaths. For example:

```
<build>
  [...]
  <plugins>
      [...]
      <plugin>
          <groupId>org.openl.rules</groupId>
          <artifactId>openl-maven-plugin</artifactId>
          <version>${openl.rules.version}</version>
          <configuration>
              <!-- Project name. -->
              <projectName>OpenL Rules Simple Project</projectName>
              <!-- Project's classpath. -->
              <classpaths>
                <param>.</param>
              </classpaths>
              <!-- OpenL project includes one or more modules. -->
              <generateInterfaces>
                  <generateInterface>
                      <displayName>Template Rules</displayName>
                      <targetClass>template.Wrapper</targetClass>
                      <!-- Rules root document. Usually excel file on file system. -->
                      <srcFile>src/main/openl/rules/TemplateRules.xls</srcFile>
                  </generateInterface>
              </generateInterfaces>
          </configuration>
          <executions>
              <execution>
                  <goals>
                      <goal>generate</goal>
                      <goal>compile</goal>
                      <goal>test</goal>
                  </goals>
              </execution>
          </executions>
      </plugin>

  </plugins>
  [...]
</build>
```

More configuration options you can find on openl:generate goal page.

8 Generate a project with working example

8.1 Create a project with working example of OpenL Maven Plugin usage

OpenL Tablets has archetype which can be used to create simple OpenL Rules project which contains an example of OpenL Maven Plugin usage. Execute in command line the following command:

```
mvn archetype:generate
```

Maven runs archetype console wizard. Select `openl-simple-project-archetype` menu item. Follow the instructions on the screen to complete all steps of the project creation wizard. After all steps are done you should have new maven-based project on the file system. It's an OpenL Rules project which has one module with simple rules. Execute in command line the following command from the root of the project folder to compile the project:

```
mvn install
```

After that, you'll find in the `target` folder:

1. A zip file (with "-deployable" suffix) for importing a project to WebStudio. You can find more information in OpenL Tablets WebStudio User Guide.
2. A zip file (with "-runnable" suffix) that can be executed after unpacking. It demonstrates how OpenL rules can be invoked from java code.
3. A jar file that contains only compiled java classes.