# OpenL Tablets and Activiti Integration Guide

Release 5.25

**Document number:** TP_OpenL_Activiti_IG_2.3_LSh

Revised: 03-07-2022

# Table of Contents

# 1 Preface

This preface is an introduction to the *OpenL Tablets and Activiti Integration Guide*. This guide contains information about OpenL Tablets integration with Activiti. Starting with OpenL Tablets 5.17.0, some modules were added for Activiti integration proposes.

The following topics are included in this preface:



## 1.1 Audience

This guide is mainly targeted at software developers who set up usage of the OpenL Tablets rules in Activiti process definitions.

## 1.2 Related Information

The following table lists sources of information related to contents of this guide:

| **Related information** | |
|---|---|
| **Title** | **Description** |
| http://activiti.org/ | Activiti official website. |
| http://openl-tablets.org/ | OpenL Tablets open source project website. |

## 1.3 Typographic Conventions

The following styles and conventions are used in this guide:

| **Typographic styles and conventions** | |
|---|---|
| **Convention** | **Description** |
| **Bold** | • Represents user interface items such as check boxes, command buttons, dialog boxes, drop-down list values, field names, menu commands, menus, option buttons, perspectives, tabs, tooltip labels, tree elements, views, and windows.<br>• Represents keys, such as **F9** or **CTRL+A.**<br>• Represents a term the first time it is defined. |
| `Courier` | Represents file and directory names, code, system messages, and command-line commands. |
| **`Courier Bold`** | Represents emphasized text in code. |
| Select **File > Save As** | Represents a command to perform, such as opening the **File** menu and selecting **Save As.** |
| *Italic* | • Represents any information to be entered in a field.<br>• Represents documentation titles. |
| < > | Represents placeholder values to be substituted with user specific values. |

| Typographic styles and conventions | |
|---|---|
| **Convention** | **Description** |
| Hyperlink | Represents a hyperlink. Clicking a hyperlink displays the information topic or external source. |
| ***[name of guide]*** | Reference to another guide that contains additional information on a specific feature. |

# 2 Maven Dependencies

The following table describes modules included into OpenL Tablets for Activiti integration:

| **Modules added for OpenL Tablets and Activiti integration** | |
|---|---|
| **Module** | **Description** |
| `org.openl.rules:org.openl.rules.activiti` | Contains integration features that do not use the OpenL Rule Service functionality. |
| `org.openl.rules:org.openl.rules.ruleservice.activiti` | Contains integration features that use OpenL Rule Service functionality. |

**Example:**

```
<dependency>
            <groupId>org.openl.rules</groupId>
            <artifactId>org.openl.rules.activiti</artifactId>
            <version>X.X.X</version>
</dependency>
<dependency>
            <groupId>org.openl.rules</groupId>
            <artifactId>org.openl.rules.ruleservice.activiti</artifactId>
            <version>X.X.X</version>
</dependency>
```

**Note:** Advised approach is to use OpenL Rule Service functionality. Only one of the dependencies above needs to be added to a project depending on a chosen approach.

# 3 OpenL Rule Service Integration

This section describes how to use OpenL Tablets Rule Service integration via Spring Integration feature in Activiti and it uses full OpenL Tablets Rule Service functionality. That is, this type of integration does not use resources added into Activiti deployment as a rule project; instead, it uses OpenL Tablets repository as the OpenL Tablets rule projects storage.

To support this functionality in Activiti process definitions, add the `openl-ruleservice-activiti-beans.xml` bean configuration into application Spring context definition.

An example is as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                  http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
       <bean id="processEngineConfiguration"
              class="org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration">
       </bean>

       <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
              <property name="processEngineConfiguration" ref="processEngineConfiguration" />
       </bean>
       <import resource="classpath:openl-ruleservice-activiti-beans.xml" />
</beans>
```

This configuration contains a bean named `openLEngine` that can be used in process definitions as a part of Spring Integration feature in Activiti. The bean contains a method with following signature:

```
public ResultValue execute(String serviceName, String methodName, Object... args) throws
Exception;
```

| **`execute` method arguments** | |
|---|---|
| **Method argument name** | **Description** |
| serviceName | OpenL Tablets service name in a repository. |
| methodName | Rule method name from a resource. |
| args | Rule method arguments. |

This method returns the `org.openl.rules.activiti.spring.result.ResultValue` class designed to simplify usage of invocation result from UEL in Activiti process definition. For more information on using this class, see Spring Integration.

An example of using the `openLRules` bean is as follows:

```
<sequenceFlow id='flow3' sourceRef='theStart' targetRef='theTask2'>
       <conditionExpression xsi:type="tFormalExpression">
              <![CDATA[${openLEngine.execute('datasource_Tutorial1', 'DriverPremium1',
              driverAge, driverMaritalStatus).toDouble() <= 400}]]>
       </conditionExpression>
</sequenceFlow>
```

Or:

```
<serviceTask id="task" activiti:expression="${openLEngine.execute('datasource_Tutorial1',
'DriverPremium1', driverAge, driverMaritalStatus).asDouble().set(execution,
'resultVariable')}" />
```

# 4 Deploying OpenL Tablets Rules in Activiti

To use OpenL Tablets rules in Activiti process definition without OpenL Tablets Rule Service, add the rules into Activiti deployment as a resource.

An example is as follows:

```
processEngine.getRepositoryService()
            .createDeployment()
            .addClasspathResource("activiti-definition.bpmn20.xml")
            .addClasspathResource("openl-rules.xls")
            .deploy();
```

The XLS, XLSX, and ZIP resource formats are supported for the OpenL Tablets rules. If a ZIP archive is used as the OpenL Tablets rules resource, the content of this archive is used as an OpenL Tablets project. In this case, the `rules.xml` configuration file is supported if it exists in the root of the archive folder.

**Note:** The current implementation of Activiti integration does not support OpenL project dependencies.

# 5 Using OpenL Tablets Rules

The `org.openl.rules.activiti.MethodInvokeResourceServiceTask` class added into the Activiti integration module implements the `JavaDelegate` interface from the Activiti framework and can be used as the Activiti service task. This class is designed to invoke the OpenL Tablets rule and stores result as an execution variable in Activiti execution context. To use this implementation, define several extension elements for a service task. For more information on a service task, see Activiti documentation.

| **`MethodInvokeResourceTask` supported elements** | | |
|---|---|---|
| **Element name** | **Required** | **Description** |
| resource | Yes | Deployment resource name for OpenL Tablets rules. |
| method | Yes | Method with its name to be used for rule execution. |
| resultVariable | Yes | Result variable name. The rule execution result is stored in the execution context with this defined name. |
| provideRuntimeContext | No | Support runtime context for OpenL Tablets rules. |
| module | No | Module from a project. This parameter can be used in multi-module projects to use the single module compilation approach. For more information on single module compilation approach, see OpenL Tablets documentation. |

If the method requires parameters, the system searches for parameters in Activiti execution variables with the same name as a parameter name used in OpenL Tablets rules. The `MethodInvokeResourceServiceTask` implementation tries to apply OpenL Tablets casts for execution context variables if the variable with required name is found but the variable type differs from the required OpenL rules parameter type. If variable cannot be found in context, the null value is used as a rule parameter.

**Note:** `MethodInvokeResourceServiceTask` cannot be used if the OpenL Tablets resource contains more than one rule method with the same method name (overloaded rules).

**Example:**

```
<serviceTask id="openLServiceTask" name="OpenL Service Task"
activiti:class="org.openl.rules.activiti.MethodInvokeResourceServiceTask">
            <extensionElements>
                        <activiti:field name="resource">
                                    <activiti:string>openl-rules.xls</activiti:string>
                        </activiti:field>
                        <activiti:field name="method">
                                    <activiti:string>rule1</activiti:string>
                        </activiti:field>
                        <activiti:field name="resultVariable">
                                    <activiti:string>resultVariable</activiti:string>
                        </activiti:field>
            </extensionElements>
</serviceTask>
```

If OpenL Tablets rules need to be used with the runtime context support feature, `MethodInvokeResourceServiceTask` supports using Activiti variables as runtime context properties.

To invoke the overloaded rules method or add additional logic to a service task, create service task implementation using `org.openl.rules.activiti.AbstractOpenLResourceServiceTask` designed for this purpose. The following table describes methods of this class that can be used in the extended class:

| **`AbstractOpenLResourceServiceTask` methods** | |
|---|---|
| **Method name** | **Description** |
| isProvideRuntimeContext | Returns true value if runtime context is supported for the OpenL Tablets resource. |
| getSimpleProjectEngineFactory | Returns factory used for OpenL Tablets resource compilation. |
| getInstance | Returns the OpenL instance object. |
| getInterfaceClass | Returns the OpenL Tablets instance interface class. |

**Example:**

```
public class SimpleOpenLServiceTask extends AbstractOpenLResourceServiceTask<Object> {
    @Override
    public void execute(DelegateExecution execution) throws Exception {
        String driverAge = (String) execution.getVariable("driverAge");
        String driverMatrialStatus = (String) execution.getVariable("driverMaritalStatus");
        Object instance = getInstance(execution);

        Class<?> clazz = getSimpleProjectEngineFactory(execution).getInterfaceClass();
        Method method = clazz.getMethod("DriverPremium1", String.class, String.class);

        DoubleValue result = (DoubleValue) method.invoke(instance, new Object[] { driverAge,
driverMatrialStatus });

        execution.setVariable("resultVariable", result.doubleValue());
    }
}
```

If a static interface exists for OpenL Tablets rules, this implementation can be rewritten with generic types as follows:

```
public interface RulesInterface {
    DoubleValue DriverPremium1(String driverAge, String driverMaritalStatus);
}

public class SimpleOpenLServiceTask extends AbstractOpenLResourceServiceTask<RulesInterface>
{

    @Override
    public void execute(DelegateExecution execution) throws Exception {
        String driverAge = (String) execution.getVariable("driverAge");

        String driverMatrialStatus = (String) execution.getVariable("driverMaritalStatus");
        RulesInterface instance = getInstance(execution);

        DoubleValue result = instance.DriverPremium1(driverAge, driverMatrialStatus);

        execution.setVariable("resultVariable", result);
    }
}
```

# 6 Spring Integration

This section describes how to use OpenL Tablets rules via Spring Integration feature in Activiti. Activiti Spring Integration feature enables using beans from Spring context in process definitions via Unified Expression Language (UEL).

To enable OpenL Tablets rules support in Activiti process definitions, add the `openl-activiti-beans.xml` bean configuration into application Spring context definition.

Spring configuration example is as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="processEngineConfiguration"
      class="org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration">
         <property name="eventListeners">
            <list>
               <bean class="org.openl.rules.activiti.spring.OpenLResourcesHandleListener" />
            </list>
         </property>
   </bean>
   <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
      <property name="processEngineConfiguration" ref="processEngineConfiguration" />
   </bean>
       <import resource="classpath:openl-activiti-beans.xml" />
</beans>
```

This bean configuration contains a bean named `openLRules`. This bean can be used in process definitions as a part of Spring Integration feature in Activiti. The bean contains a method with following signature:

```
public ResultValue execute(DelegateExecution execution, String resource, String methodName,
            Object... args) throws Exception;
```

| **`execute` method arguments** | |
|---|---|
| **Method argument name** | **Description** |
| execution | Activiti execution. |
| resource | Name of the OpenL Tablets resource in deployment. |
| methodName | Rule method name from a resource. |
| args | Rule method arguments. |

This method returns `org.openl.rules.activiti.spring.result.ResultValue` class designed to simplify usage of invocation result from UEL in Activiti process definition. The result easy can be casted to different types by using the following methods:

| **Methods for casting invocation result to different types** | |
|---|---|
| **Method name** | **Description** |
| asByte(), asInt(), asLong(), asDouble(), asFloat(), asString(), asBoolean() | Converts ResultValue to the new ResultValue that stores result as Byte, Integer, Long, Float, Double, String, or Boolean value. |
| toByte(), toInt(), toLong(), toDouble(), toFloat(), toString(), toBoolean() | Converts ResultValue to Byte, Integer, Long, Float, Double, String, or Boolean value. |

| Methods for casting invocation result to different types | |
|---|---|
| **Method name** | **Description** |
| value() | Returns the result value. |
| set(DelegateExecution, String variableName), set(DelegateExecution, String variableName, boolean fetchAllVariables), setLocal(DelegateExecution, String variableName), setLocal(DelegateExecution, String variableName, boolean fetchAllVariables), | Sets the result value into execution as a defined variable. |

If OpenL Tablets rules are compiled with runtime context support feature, pass runtime context as the first method argument. To build runtime context automatically from Activiti context variables with the same names as runtime context properties, use the following method of the `openLRules` bean:

```
protected IRulesRuntimeContext buildRuntimeContext(DelegateExecution execution);
```

An example of using the `openLRules` bean is as follows:

```
<sequenceFlow id='flow2' sourceRef='theStart' targetRef='theTask1'>
       <conditionExpression xsi:type="tFormalExpression">
              <![CDATA[${openLRules.execute(execution, 'openl-rules.xls', 'DriverPremium1',
              driverAge, driverMaritalStatus).toDouble() > 400}]]>
       </conditionExpression>
</sequenceFlow>
```

Alternatively, use the `openLRules` bean as follows:

```
<serviceTask id="task" activiti:expression="${openLRules.execute(execution, 'openl-
rules.xls', 'DriverPremium1', driverAge, driverMaritalStatus).asDouble().set(execution,
'resultVariable')}" />
```

Build runtime context usage example is as follows:

```
<serviceTask id="task" activiti:expression="${openLRules.execute(execution, 'openl-
rules.xls', 'DriverPremium1', openLRules.buildRuntimeContext(execution), driverAge,
driverMaritalStatus).asDouble().set(execution, 'resultVariable')}" />
```