**OpenL Tablets Integration with Activiti Guide**

This guide contains information about OpenL Tablets integration with Activiti. Starting from OpenL Tablets 5.17.0 version a few modules was added for Activiti integration proposes.

**Maven dependencies**

There are two modules were added into OpenL Tablets for Activiti integration. The first module (org.openl.rules:org.openl.rules.activiti) contains integration features that doesn’t use OpenL Rule Service functionality. This functionality was extracted into second module (org.openl.rules:org.openl.rules.ruleservice.activiti).

<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>

**How to deploy OpenL Tablets rules?**

If you want to use OpenL Tablet rules in Activiti process definition, the first you should add OpenL rules into Activiti deployment. OpenL Tablet rules should be added to deployment as resource.

For example:

processEngine.getRepositoryService()

.createDeployment()

.addClasspathResource("activiti-definition.bpmn20.xml")

.addClasspathResource("openl-rules.xls")

.deploy();

XLS, XLSX and ZIP resource format supported for OpenL Tablets rules. If you use ZIP archive as OpenL Tablets rules resource, then the content of this archive is used as OpenL Tablets project (rules.xml configuration file is supported, if it exists in the root of archive folder).

**Note:** Current implementation of Activiti Integration doesn’t support project dependencies feature. This feature will be added in future.

**How to use OpenL Tablets rules?**

org.openl.rules.activiti.MethodInvokeResourceServiceTask class was added into Activiti integration module. This class implement JavaDelegate interface from Activiti framework and can be used as Service Task. This class is designed to invoke OpenL Tablets rule and store result as execution variable in Activiti execution context. To use this implementation you have to define a few extension elements for Service Task. Please, refer to Activiti documentation for more information about Service Task.

| Element name | Required | Description |
| --- | --- | --- |
| resource | Yes | Define deployment resource name for OpenL Tablets rules. |
| method | Yes | The method with this name will be used for rule execution. |
| resultVariable | Yes | Define a result variable name. The rule execution result will be store in execution context with this defined name. |
| provideRuntimeContext | No | Is support runtime context for OpenL Tablets rules. |
| module | No | Use this module from project. This parameter can be used in multi-module projects in case to use single module compilation approach. Please, refer OpenL Tablets documentation for more details. |

If requested method requires method parameters, the system search parameters in Activiti execution variables with the same name as parameter name used in OpenL Tables rules. The MethodInvokeResourceServiceTask implementation tries to apply OpenL casts for execution context variables, if variable with required name was found, but variable type is differ from required OpenL rules parameter type. If variable can’t be found in context then null values is used as rule parameter.

**Note:** MethodInvokeResourceServiceTask can’t be used OpenL Tablets resource contains more than one rule method with defined method name.

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 Tablet rules were compiled with runtime context support feature, then MethodInvokeResourceServiceTask supports using Activiti variables as runtime context properties.

If you want to invoke overloaded rules method or add additional logic to Service Task, you have to create your own Service Task implementation. org.openl.rules.activiti. AbstractOpenLResourceServiceTask is designed for this propose. This class has following method that can be used in extended class:

| Method name | Description |
| --- | --- |
| isProvideRuntimeContext | This method returns true value, if runtime context supported for OpenL Tablets resource. |
| getSimpleProjectEngineFactory | Returns factory that was used for OpenL Tablets resource compilation |
| getInstance | Return OpenL instance object |
| getInterfaceClass | Returns 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 static interface exists for OpenL rules this implementation can be rewritten with generic types:

**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);

}

}

**Spring Integration**

This section of guide contains information about how to use OpenL Tablet rules via Spring Integration feature in Activiti. Activiti Spring Integration feature enable using beans from Spring context in process definitions via UEL (Unified Expression Language).

For support OpenL Tablets rules in Activiti process definitions, openl-activiti-beans.xml bean configuration should be added into application spring context definition:

Spring configuration example:

<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 with name openLRules. This bean can be used in process definitions as part of Spring Integration feature in Activiti. The bean contains one method with following signature:

**public** ResultValue execute(DelegateExecution execution, String resource, String methodName,

Object... args) **throws** Exception;

| Method argument name | Description |
| --- | --- |
| execution | Activiti execution |
| resource | The name of OpenL Tablets resource in deployment |
| methodName | Rule method name from resource |
| args | Rule method arguments |

This method returns org.openl.rules.activiti.spring.result.ResultValue class. This class was designed for easy to use invocation result from UEL in Activiti process definition. The result easy can be casted to different type by following methods:

| Method name | Description |
| --- | --- |
| asByte(), asInt(), asLong(), asDouble(), asFloat(), asString(), asBoolean() | Converts ResultValue to the new ResultValue that store 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 |
| value() | Returns result value |
| set(DelegateExecution, String variableName), set(DelegateExecution, String variableName, boolean fetchAllVariables),  setLocal(DelegateExecution, String variableName), setLocal(DelegateExecution, String variableName, boolean fetchAllVariables), | Set result value into execution as defined variable |

If OpenL Tablet rules were compiled with runtime context support feature, then you have to pass runtime context as first method argument. If you want to build runtime context automatically from Activiti context variables with the same names as runtime context properties, then use following method from openLRules bean:

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

See following example of using openLRules bean:

<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>

Or:

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

Build runtime context usage example:

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

**OpenL Rule Service Integration**

This type of integration is similar to Spring Integration, but uses OpenL Tablets uses full OpenL Rule Service funtionality. This means that this type of integration doesn’t use added resources into Activiti deployment as rule project, it uses OpenL Tablets repository as OpenL rule projects storage.

For support this functionality in Activiti process definitions, openl-ruleservice-activiti-beans.xml bean configuration should be added into application spring context definition:

For example:

<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 with name openLEngine. This bean can be used in process definitions as part of Spring Integration feature in Activiti. The bean contains one method with following signature:

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

| Method argument name | Description |
| --- | --- |
| serviceName | OpenL service name in repository |
| methodName | Rule method name from resource |
| args | Rule method arguments |

This method returns org.openl.rules.activiti.spring.result.ResultValue class. This class was designed for easy to use invocation result from UEL in Activiti process definition. Please, read previous section for more details.

See following example of using openLRules bean:

<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')}"* />