



EIS GROUPTM

**Developers Guide
OpenL Tablets BRMS
Release 5.15**

Document number: TP_OpenL_DG_3.0_LSh

Revised: 08-12-2015



OpenL Tablets Documentation is licensed under a [Creative Commons Attribution 3.0 United States License](https://creativecommons.org/licenses/by/3.0/us/).

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Preface..... | 4 |
| 1.1 | Audience..... | 4 |
| 1.2 | Related Information | 4 |
| 1.3 | Typographic Conventions | 4 |
| 2 | Introducing OpenL Tablets | 6 |
| 2.1 | What Is OpenL Tablets?..... | 6 |
| 2.2 | Basic Concepts..... | 6 |
| | Rules..... | 7 |
| | Tables | 7 |
| | Projects | 7 |
| | Wrapper | 7 |
| | Execution Mode for OpenL Project | 7 |
| 2.3 | System Overview | 8 |
| 2.4 | Quick Start with OpenL Tablets | 8 |
| 3 | OpenL Tablets Rules Projects | 10 |
| 3.1 | OpenL Rules Project | 10 |
| 3.2 | Rules Project Descriptor | 10 |
| | Quick Overview | 10 |
| | Descriptor Elements..... | 11 |
| 3.3 | Project Resolving | 12 |
| 3.4 | How to Start with OpenL Rules Project | 13 |
| | Creating a Project Using the Maven Archetype | 13 |
| | Creating a Project in OpenL Tablets WebStudio | 14 |
| | Creating a Project Manually | 15 |
| | Editing Rules..... | 15 |
| | Using OpenL Tablets Rules from Java Code | 15 |
| | Handling Data and Data Types in OpenL Tablets | 21 |
| 3.5 | Customizing Table Properties..... | 23 |
| | Dispatching Table Properties | 23 |
| 3.6 | Validation for Tables..... | 24 |
| | Validators | 24 |
| 3.7 | Module Dependencies..... | 25 |
| | Classloaders | 25 |
| 3.8 | Peculiarities of OpenL Tablets Implementation | 26 |
| | Lookup Tables Implementation Details..... | 27 |
| | Range Types Instantiation | 27 |

1 Preface

This preface is an introduction to this *OpenL Tablets Developer Guide*.

The following topics are included in this preface:

- [Audience](#)
- [Related Information](#)
- [Typographic Conventions](#)

1.1 Audience

This guide is mainly intended for developers who create applications employing the table based decision making mechanisms offered by OpenL Tablets technology. However, business analysts and other users can also benefit from this guide by learning the basic OpenL Tablets concepts described herein.

Basic knowledge of Java, Ant, and Microsoft Excel is required to use this guide effectively.

1.2 Related Information

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

| Related information | |
|---|---|
| Title | Description |
| [OpenL Tablets WebStudio User Guide] | Document describing OpenL Tablets WebStudio, a web application for managing OpenL Tablets projects through a web browser. |
| http://openl-tablets.sourceforge.net/ | 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 | <ul style="list-style-type: none">• 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.• Represents keys, such as F9 or CTRL+A.• 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 . |
| <i>Italic</i> | <ul style="list-style-type: none">• Represents any information to be entered in a field.• 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. |
| <i>[name of guide]</i> | Reference to another guide that contains additional information on a specific feature. |

2 Introducing OpenL Tablets

This chapter introduces OpenL Tablets and describes its main concepts.

The following topics are included in this chapter:

- [What Is OpenL Tablets?](#)
- [Basic Concepts](#)
- [System Overview](#)
- [Quick Start with OpenL Tablets](#)

2.1 What Is OpenL Tablets?

OpenL Tablets is a business rules management system and business rules engine based on tables presented in Excel documents. Using unique concepts, OpenL Tablets facilitates treating business documents containing business logic specifications as executable source code. Since the format of tables used by OpenL Tablets is familiar to business users, OpenL Tablets bridges a gap between business users and developers, thus reducing costly enterprise software development errors and dramatically shortening the software development cycle.

In a very simplified overview, OpenL Tablets can be considered as a table processor that extracts tables from Excel documents and makes them accessible from application.

The major advantages of using OpenL Tablets are as follows:

- OpenL Tablets removes the gap between software implementation and business documents, rules, and policies.
- Business rules become transparent to developers.
For example, decision tables are transformed into Java methods or directly into web service methods. The transformation is performed automatically.
- OpenL Tablets verifies syntax and type errors in all project document data, providing convenient and detailed error reporting. OpenL Tablets is able to directly point to a problem in an Excel document.
- OpenL Tablets provides calculation explanation capabilities, enabling expansion of any calculation result by pointing to source arguments in the original documents.
- OpenL Tablets enables users to create and maintain tests to insure reliable work of all rules.
- OpenL Tablets provides cross-indexing and search capabilities within all project documents.
- OpenL Tablets provides full rules lifecycle support through its business rules management applications.
- OpenL Tablets supports the `.xls` and `.xlsx` file formats.

2.2 Basic Concepts

This section describes the basic concepts of OpenL Tablets:

- [Rules](#)
- [Tables](#)
- [Projects](#)
- [Wrapper](#)
- [Execution Mode for OpenL Project](#)

Rules

In OpenL Tablets, a **rule** is a logical statement consisting of conditions and actions. If a rule is called and all its conditions are true, then the corresponding actions are executed. Basically, a rule is an IF-THEN statement. The following is an example of a rule expressed in human language:

If a service request costs less than 1,000 dollars and takes less than 8 hours to execute, then the service request must be approved automatically.

Instead of executing actions, rules can also return data values to the calling program.

Tables

Basic information OpenL Tablets deals with, such as rules and data, is presented in **tables**. Different types of tables serve different purposes. For detailed information on table types, see the [OpenL Tablets Reference Guide](#), the *Table Types* section.

Projects

An OpenL Tablets **project** is a container of all resources required for processing rule related information. Usually, a project contains Excel files and Java code. For detailed information on projects, see the [OpenL Tablets Reference Guide](#), chapter *working with Projects*.

There can be situations where OpenL Tablets projects are used in the development environment but not in production, depending on the technical aspects of a solution.

Wrapper

A **wrapper** is a Java object that exposes rule tables via Java methods and data tables as Java objects and allows developers to access table information from code. Wrappers are essential for solutions where compiled OpenL Tablets project code is embedded in solution applications. If tables are accessed through web services, client applications are not aware of wrappers but they are still used on the server.

For more information about wrappers, see [Using OpenL Tablets rules from Java Code](#).

Execution Mode for OpenL Project

Execution mode for OpenL project is a light weight compilation mode that enables only evaluating of rules; but editing, tracing and search are not available. Since the Engine will not load test tables and keep debug information in memory in this mode, memory consumption is up to 5 times less than for debug mode.

By default, the execution mode (`exectionMode=true`) is used in OpenL Tablets Web Services.

The debug mode (`exectionMode=false`) is used by default in WebStudio.

Flag indicating required mode is introduced in Runtime API and in wrappers.

To compile OpenL project in execution mode:

- In case OpenL high level API (instantiation strategies) is used, an execution mode can be defined in constructor of the particular instantiation strategy.
- In case low level API (Engine factories) is used, an execution mode flag can be set using the `setExecutionMode(boolean)` method.

2.3 System Overview

The following diagram shows how OpenL Tablets is used by different types of users:

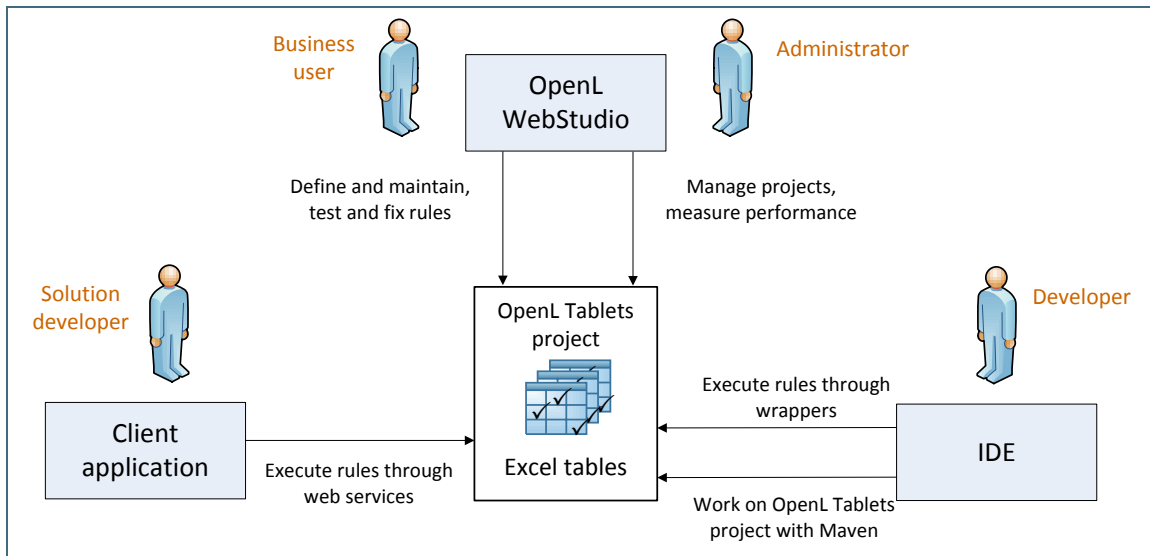


Figure 1: OpenL Tablets overview

The following is a typical lifecycle of an OpenL Tablets project:

1. A business analyst creates a new OpenL Tablets project in WebStudio. Optionally, development team may provide the analyst with a project in case of complex configuration.
The business analyst also creates correctly structured tables in Excel files based on requirements and includes them in the project. Typically, this task is performed through Excel or OpenL Tablets WebStudio in a web browser.
2. The business analyst performs unit and integration tests by creating test tables and performance tests on rules through OpenL Tablets WebStudio. As a result, fully working rules are created and are ready to be used.
3. A developer adds configuration to the project according to application needs. As alternative approach they can create a new OpenL Tablets project in their IDE via OpenL Maven Archetype and adjust it to use business user input.
4. A developer employs business rules directly through the OpenL Tablets engine or remotely through web services.
5. Whenever required, the business user updates or adds new rules to project tables. OpenL Tablets business rules management applications (OpenL Tablets WebStudio, Rules Repository, Rule Service) can be set up to provide self-service environment for business user changes.

2.4 Quick Start with OpenL Tablets

OpenL Tablets provide a few ways to create a new project. We recommend using Simple Project Maven Archetype approach for creating a new project first time or create it via WebStudio. All approaches of creating a project can be found with detailed descriptions in [How to Start With OpenL Rules Project](#).

After a project is created, a zip or Excel file for importing the project to OpenL Tablets WebStudio can be used. For detailed information on importing an existing project into WebStudio, see the [OpenL Tablets WebStudio User Guide](#).

OpenL Tablets WebStudio provides convenient UI to work with rules. However, its usage can be avoided by working with rules from IDE only using the OpenL Tablets Maven Plugin. The plugin provides compilation and testing of rules and wrapper generation support.

Also, OpenL Tablets has OpenL Tablet Demo Package. It can be downloaded on the [OpenL Tablets website](#). A demo is a zip file that contains a Tomcat with configured OpenL Tablets WebStudio and OpenL Tablets Web Services projects. It can be used to effectively start using OpenL Tablets products.

3 OpenL Tablets Rules Projects

This chapter describes how to create and use OpenL Tablets Rules projects.

The following topics are included in this chapter:

- [OpenL Rules Project](#)
- [Rules Project Descriptor](#)
- [Project Resolving](#)
- [How to start with OpenL Rules Project](#)
- [Customizing Table Properties](#)
- [Validation for Tables](#)
- [Module Dependencies](#)
- [Peculiarities of OpenL Tablets Implementation](#)

3.1 OpenL Rules Project

OpenL Rules project is a project that contains Excel files with OpenL Tablets rules and may contain a rules project descriptor. The rules project descriptor is a XML file that defines project configuration and allows setting project dependencies.

OpenL Rules Project can easily use rules from other projects via dependency functionality.

3.2 Rules Project Descriptor

A rules project descriptor is a XML file that contains information about the project and configuration details used by OpenL to load and compile the rules project. The predefined name that is used for a rules project descriptor is *rules.xml*.

This section includes the following topics:

- [Quick Overview](#)
- [Descriptor Elements](#)

Quick Overview

The following code fragment is an example of the rules project descriptor:

```
<project>
  <!-- Project name. -->
  <name>Project name</name>
  <!-- Optional. Comment string to project. -->
  <comment>comment</comment>

  <!-- OpenL project includes one or more rules modules. -->
  <modules>

    <module>
      <name>MyModule1</name>
      <type>API</type>

  <!--
                                Rules document which is usually an excel file in the project.
  -->
```

```
<rules-root path="MyModule1.xls"/>

</module>

<module>
  <name>MyModule2</name>
  <type>API</type>

<!--
      Rules document which is usually an excel file in the project.
-->
  <rules-root path="MyModule2.xls"/>
  <method-filter>
    <includes>
      <value> * </value>
    </includes>
  </method-filter>
</module>
</modules>

<dependencies>
  <dependency>
    <name>projectName</name>
    <autoIncluded>false</autoIncluded>
  </dependency>
</dependencies>
<properties-file-name-pattern>{lob}</properties-file-name-pattern>
<properties-file-name-processor>default.DefaultPropertiesFileNameProcessor</properties-file-name-processor>
<!-- Project's classpath (list of all source dependencies). -->
<classpath>
  <entry path="path1"/>
  <entry path="path2"/>
</classpath>

</project>
```

Descriptor Elements

The descriptor file contains several sections that describe the projects configuration:

- [Project Configurations](#)
- [Module Configurations](#)
- [Dependency Configurations](#)
- [Classpath Configurations](#)

Project Configurations

The project configurations are as follows:

| Project section | | |
|-----------------|----------|--|
| Tag | Required | Description |
| name | yes | Project name. String value which defines user-friendly project name. |
| comment | no | Comment for project. |
| dependency | no | Defines dependencies to projects. |
| modules | yes | Defines modules of project. Project can have one or more modules. |
| classpath | no | Project relative classpath. |

| Project section | | |
|--------------------------------|----------|---|
| Tag | Required | Description |
| properties-file-name-pattern | no | File name pattern that will be used by file name processor. File name processor adds extracted module properties from a module file name. |
| properties-file-name-processor | no | Custom implementation of <i>org.openl.rules.project.PropertiesFileNameProcessor</i> that is used instead of default implementation. |

Module Configurations

The module configurations are as follows:

| Module section | | |
|----------------|----------|---|
| Tag | Required | Description |
| name | yes/no | Module name. String value which defines user-friendly module name. <i>Note:</i> Used by WebStudio application as module display name. Not required for modules defined via wildcard. |
| type | yes | Module instantiation type. Possible values (case-insensitive): dynamic, api, static (deprecated). Defines the way of OpenL project instantiation. |
| classname | yes/no | Used together with <i>type</i> . Defines name of rules interface. Not required for <i>api</i> type. |
| method-filer | no | Filter that defines what tables will be used for interface generation. Java regular expression can be used to define a filter for multiple methods. |
| rules-root | yes/no | Used together with <i>type</i> . Defines path to the main file of rules module. Ant pattern can be used to define multiple modules via wildcard. See Ant patterns for more information. |

Dependency Configurations

The dependency configurations are as follows:

| Dependency section | | |
|--------------------|----------|---|
| Tag | Required | Description |
| name | yes | Dependency project name. |
| autoIncluded | yes | If true, all modules from dependency project will be used in this project module. If false, modules from dependency project can be used in this project as dependencies and each module will define its own list of used dependency modules. |

Classpath Configurations

The classpath configurations are as follows:

| Classpath section | | |
|-------------------|----------|---|
| Tag | Required | Description |
| entry | no | Defines path for the classpath entry (classes or jar file). |

3.3 Project Resolving

The *RulesProjectResolver* Java class resolves all OpenL projects inside the workspace. The Resolver lists all folders in the workspace and tries to detect the OpenL project by a predefined strategy. The easiest way to

initialize *RulesProjectResolver* is to use the static method *loadProjectResolverFromClassPath()* that uses **project-resolver-beans.xml** from classpath (usual Spring beans config that defines all resolving strategies and their order).

Make sure that the resolving strategies are in the correct order – some projects may be matched by several resolving strategies. By default, the resolving strategies are in the following order:

Project Descriptor Resolving Strategy

This resolving strategy is the strictest one. This strategy is based on the descriptor file (see above).

Excel File Resolving Strategy

This is the resolving strategy for the simplest OpenL project which contains only Excel files in root folder without wrappers and descriptor. Each Excel file represents a module.

3.4 How to Start with OpenL Rules Project

Firstly, an OpenL Rules project must be created. It can be done in the following ways:

- using maven archetype
- using OpenL Tablets WebStudio
- manually

See the following sections for detailed information:

- [Creating a Project Using the Maven Archetype](#)
- [Creating a Project in OpenL Tablets WebStudio](#)
- [Creating a Project Manually](#)
- [Editing Rules](#)
- [Using OpenL Tablets Rules from Java Code](#)
- [Handling Data and Data Types in OpenL Tablets](#)

Creating a Project Using the Maven Archetype

OpenL Tablets provides the Maven archetype which can be used to create a simple OpenL Rules project.

To create a project using the Maven archetype, proceed as follows:

1. Execute the following command in command line:

```
mvn archetype:generate
```

Maven runs the archetype console wizard.
2. Select the *openl-simple-project-archetype* menu item.
3. Follow with the Maven creation wizard.

After all steps are completed, a new Maven based project appears on the file system. It is an OpenL Rules project which has one module with simple rules in it.

Execute the following command in command line from the root of the project folder to compile the project:

```
mvn install
```

After executing this command, the following files can be found in the target folder:

1. zip file (with "-deployable" suffix) for importing the project to WebStudio. For more information, see [\[OpenL Tablets WebStudio User Guide\]](#).

2. zip file (with "-runnable" suffix) that can be executed after unpacking it. It demonstrates how OpenL rules can be invoked from Java code.
3. jar file that contains only compiled Java classes. This jar can be put in classpath of the project and used as a depended library.

Creating a Project in OpenL Tablets WebStudio

OpenL Tablets WebStudio allows users to create new rule projects in the Repository in one of the following ways:

- Create a rule project from template
- Create a rule project from Excel file(s)
- Create a rule project from zip archive
- Import a rule project from workspace

The following diagram explains how projects are stored in WebStudio and then deployed and used by WebServices:

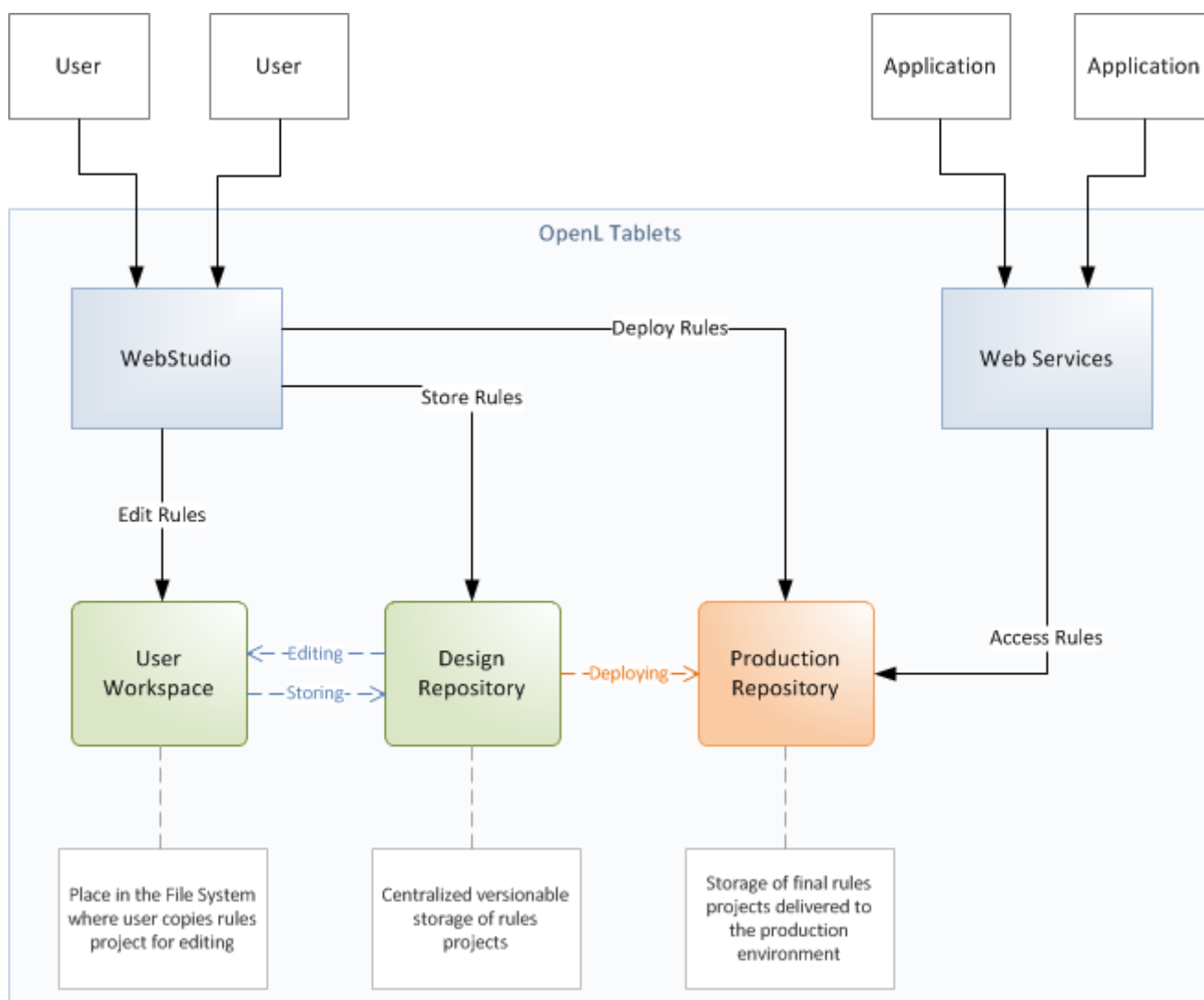


Figure 2: OpenL Tablets WebStudio and Web Services Integration

When a user starts editing a project, it is extracted from Design Repository and placed on the File System (User Workspace). The project becomes locked in Design Repository for editing by other users. After editing is finished, the user saves the project. It means that an updated version of the project is saved to Design Repository and available for editing by other users.

Web Services use separate repository instance (Production Repository in Figure 2). WebStudio can be configured to deploy complete and tested rules projects to that repository.

For more information, see the [OpenL Tablets WebStudio User Guide](#).

Creating a Project Manually

OpenL does not oblige a user to use predefined ways of project creation and provides a way to use the user's own project structure. The [Project Resolving](#) mechanism can be used as a base for the project structure definition. Depending on the resolving strategy, more or less files and folders need to be created, but several project's elements must be defined. For more details, see [OpenL Rules Project](#).

Editing Rules

After a project has been created, business rules have to be created next. It can be done using the OpenL Tablets WebStudio application or manually using MS Excel. If the simple rules project is used, there are several simple predefined rules that can be used as an example.

Using OpenL Tablets Rules from Java Code

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

This section illustrates the creation of a wrapper for a **Simple** project in IDE. There is only one rule **hello1** in the **Simple** project by default.

| Rules void hello1(int hour) | | | |
|-----------------------------|-------------|-------------|---|
| Rule | C1 | C2 | A1 |
| | min <= hour | hour <= max | System.out.println(greeting + ", World!") |
| | int min | int max | String greeting |
| Rule | From | To | Greeting |
| R10 | 0 | 11 | Good Morning |
| R20 | 12 | 17 | Good Afternoon |
| R30 | 18 | 21 | Good Evening |
| R40 | 22 | 23 | Good Night |

Figure 3: The hello1 rule table

Proceed as follows:

1. In the project `src` folder, create an interface as follows:

```
public interface Simple {  
    void hello1(int i);  
}
```

2. Create a wrapper object as follows:

```
import static java.lang.System.out;  
import org.openl.rules.runtime.RulesEngineFactory;  
  
public class Example {
```

```
public static void main(String[] args) {
    //define the interface
    RulesEngineFactory<Simple> rulesFactory =
        new RulesEngineFactory<Simple>("TemplateRules.xls",
                                       Simple.class);

    Simple rules = (Simple) rulesFactory.newInstance();
    rules.hello1(12);
}
}
```

When the class is run, it executes and displays **Good Afternoon, World!**

The interface can be generated by OpenL Tablets in runtime if the developer does not define it when initializing the rule engine factory. In this case rules can be executed via reflection.

The following example illustrates using a wrapper with a generated interface in runtime:

```
public static void callRulesWithGeneratedInterface(){
    // Creates new instance of OpenL Rules Factory
    RulesEngineFactory<?> rulesFactory =
new RulesEngineFactory<Object>("TemplateRules.xls");
    //Creates new instance of dynamic Java Wrapper for our lesson
    Object rules = rulesFactory.newInstance();

    //Get current hour
    Calendar calendar = Calendar.getInstance();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);

    Class<?> clazz = rulesFactory.getInterfaceClass();

    try{
        Method method = clazz.getMethod("hello1", int.class);
        out.println("** Executing OpenL rules...\n");
        method.invoke(rules, hour);
    }catch(NoSuchMethodException e){
    }catch (InvocationTargetException e) {
    }catch (IllegalAccessException e) {
    }
}
}
```

This section includes the following topics:

- [Using OpenL Tablets Rules with the Runtime Context](#)
- [Using OpenL Tablets Projects from Java Code](#)
- [Accessing a Test Table from Java Code](#)
- [Generating Java Classes from Datatype Tables](#)

Using OpenL Tablets Rules with the Runtime Context

This section describes the use of the runtime context for dispatching versioned rules by dimension properties values.

For example, two rules are overloaded by dimension properties. Both rules have the same name.

The first rule, covering an Auto line of business, is as follows (pay attention to the line with the lob property):

| Rules void hello1(int hour) | | | |
|-----------------------------|------------------|-------------|---|
| properties | createdOn | 4/7/10 | |
| | createdBy | LOCAL | |
| | lob | Auto | |
| Rule | C1 | C2 | A1 |
| | min <= hour | hour <= max | System.out.println(greeting + ", World!") |
| | int min | int max | String greeting |
| Rule | From | To | Greeting |
| R10 | 0 | 11 | Good Morning |
| R20 | 12 | 17 | Good Afternoon |
| R30 | 18 | 21 | Good Evening |
| R40 | 22 | 23 | Good Night |

Figure 4: The auto rule

The second rule, covering a Home line of business, is as follows:

| Rules void hello1(int hour) | | | |
|-----------------------------|-------------------|-------------|---|
| properties | modifyOn | 4/7/10 | |
| | modifiedBy | LOCAL | |
| | lob | Home | |
| Rule | C1 | C2 | A1 |
| | min <= hour | hour <= max | System.out.println(greeting + ",Guys!") |
| | int min | int max | String greeting |
| Rule | From | To | Greeting |
| R10 | 0 | 11 | It is Mornig |
| R20 | 12 | 17 | It is Afternoon |
| R30 | 18 | 21 | It is Evening |
| R40 | 22 | 23 | It is Night |

Figure 5: The home rule

A wrapper enables the user to define which of these rules must be executed:

```
// Getting runtime environment which contains context
IRuntimeEnv env = ((IEngineWrapper) rules).getRuntimeEnv();

// Creating context
IRulesRuntimeContext context = new DefaultRulesRuntimeContext();
env.setContext(context);
// define context
context.setLob("Home");
```

As a result, the code of the wrapper with the run-time context resembles the following:

```
import static java.lang.System.out;

import org.openl.rules.context.DefaultRulesRuntimeContext;
import org.openl.rules.context.IRulesRuntimeContext;
import org.openl.rules.runtime.RulesEngineFactory;
import org.openl.runtime.IEngineWrapper;
import org.openl.vm.IRuntimeEnv;
public class ExampleOfUsingRuntimeContext {

    public static void main(String[] args) {
        //define the interface
        RulesEngineFactory<simple> rulesFactory = new
RulesEngineFactory<Simple>("TemplateRules.xls", Simple.class);
        Simple rules = (Simple) rulesFactory.newInstance();
```

```
// Getting runtime environment which contains context
IRuntimeEnv env = ((IEngineWrapper) rules).getRuntimeEnv();
// Creating context (most probably in future, the code will be different)
IRulesRuntimeContext context = RulesRuntimeContextFactory.
buildRulesRuntimeContext();          env.setContext(context);
context.setLob("Home");
rules.hello1(12);

    }
}
```

Run this class. In the console, ensure that the rule with **lob = Home** was executed. With the input parameter **int = 12**, the **It is Afternoon, Guys** phrase is displayed.

Using OpenL Tablets Projects from Java Code

OpenL Tablets projects can be instantiated via `SimpleProjectEngineFactory`. This factory is designed to be created via `SimpleProjectEngineFactoryBuilder`. A builder has to be configured. The main builder method is `setProject(String location)`. The project location folder has to be specified via this method.

The following example instantiates the OpenL Tablets project:

```
ProjectEngineFactory<Object> projectEngineFactory = new
SimpleProjectEngineFactory.SimpleProjectEngineFactoryBuilder<Object>().setProject(<project
location>) .build();
Object instance = projectEngineFactory.newInstance();
```

The above example instantiates the OpenL Tablets project generated in runtime interface. A method from instantiated project can be invoked via reflection mechanism. `ProjectEngineFactory` returns generated interface via the `getInterfaceClass()` method.

If a static interface needs to be used, the interface must be specified in `SimpleProjectEngineFactoryBuilder`. The following example illustrates how to instantiate a project with a static interface.

```
SimpleProjectEngineFactory<SayHello> simpleProjectEngineFactory = new
SimpleProjectEngineFactoryBuilder<SayHello>().setProject(<project location>)
    .setInterfaceClass(SayHello.class)
    .build();
SayHello instance = simpleProjectEngineFactory.newInstance();
```

`SimpleProjectEngineFactoryBuilder` has additional methods to configure an engine factory. For example, the `setWorkspace()` method defines a project workspace for dependent projects resolving. The execution mode can be changed via the `setExecutionMode()` method, by default the runtime execution mode is enabled. If the instance class needs to provide runtime context, it must be specified via `setProvideRuntimeContext(true)`.

OpenL Tablets WebStudio supports compilation of a module from project in single mode. A module can be compiled from project in single module via `setModule(String moduleName)`. If this method is used with the module name "single mode", compilation for this module will be used from the project.

Accessing a Test Table from Java Code

Test results can be accessed through the test table API. For example, the following code fragment executes all test runs in a test table called **insuranceTest** and displays the number of failed test runs:

```
RulesEngineFactory<?> rulesFactory = new RulesEngineFactory<?>("Tutorial_1.xls");
IOpenClass openClass = rulesFactory.getCompiledOpenClass();
IRuntimeEnv env = SimpleVMFactory.buildSimpleVM().getRuntimeEnv();
Object target = openClass.newInstance(env);
IOpenMethod method = openClass.getMatchingMethod("testMethodName", testMethodParams);
TestUnitsResults res = (TestUnitsResults) testMethod.invoke(engine, new Object[0], env);
```

Generating Java Classes from Datatype Tables

Some rules require complex data models as input parameters. Developers have to generate classes for each datatype defined in an Excel file for using them in a static interface as method arguments. The static interface can be used in engine factory. For information on how to create and use a wrapper, see [Using OpenL Tablets rules from Java Code](#).

Note: Datatype is an OpenL table of the Datatype type that is created by a business user and it defines a custom data type. Using these data types inside the OpenL Tablets rules is recommended as the best practice. For detailed information on datatypes, see [\[OpenL Tablets Reference Guide\]](#), Datatype Table section.

To generate datatype classes, proceed as follows:

1. In Maven:
 - a. Configure the OpenL Maven plugin as described in [Configuring the OpenL Maven Plugin](#).
 - b. Run the Maven script.
2. In Ant:
 - a. Configure the Ant task file as described in [Configuring the Ant Task File](#).
 - b. Execute the Ant task file.

Configuring the OpenL Maven Plugin

To generate an interface for rules and datatype classes defined in the MS Excel file, add the following Maven configuration to the *pom.xml* file:

```
<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 project descriptor (rules.xml) is generated on each Maven run on generate-sources phase.

Each `<generateInterface>` section has a number of parameters. The following table describes the `<generateInterface>` section parameters.

| <generateInterface> section parameters | | | |
|---|-------------|-----------------|---|
| 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 is imported by the module when the interface is generated. Default value is: null. |
| ignoreTestMethods | boolean | false | If true, test methods are not 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. |

For more configuration options, see the [OpenL Tablets Maven Plugin Guide](#).

Configuring the Ant Task File

The following is an example of the build file:

```
<project name="GenJavaWrapper" default="generate" basedir="../">
  <taskdef name="openlgen" classname="org.openl.conf.ant.JavaWrapperAntTask"/>

  <target name="generate">
    <echo message="Generating wrapper classes..." />

    <openlgen openlName="org.openl.xls" userHome="."
      srcFile="rules/Rules.xls"
      targetClass="com.exigen.claims.RulesWrapper"
      displayName="Rule datatypes"
      targetSrcDir="gen"
    >
  </openlgen>

  <openlgen openlName="org.openl.xls" userHome="."
    srcFile="rules/Data.xls"
```

```
        targetClass=" com.exigen.claims.DataWrapper"
        displayName="Data datatypes"
        targetSrcDir="gen"
    >
</openlgen>

</target>
</project>
```

When the file is executed, it automatically creates Java classes for datatypes for specified Excel files. The Ant task file must be adjusted to match contents of the specific project.

For each Excel file, an individual `<openlgen>` section must be added between the `<target>` and `</target>` tags. Each `<openlgen>` section has a number of parameters that must be adjusted. The following table describes the `<openlgen>` section parameters:

| <openlgen> section parameters | |
|--|---|
| Parameter | Description |
| <code>openlName</code> | OpenL configuration to be used. For OpenL Tablets, the following value must always be used: <code>org.openl.xls</code> |
| <code>userHome</code> | Location of user defined resources relative to the current OpenL Tablets project. |
| <code>srcFile</code> | Reference to the Excel file for which a wrapper class must be generated. |
| <code>targetClass</code> | Full name of the wrapper class to be generated. |
| <code>displayName</code> | End user oriented title of the file that appears in OpenL Tablets WebStudio. |
| <code>targetSrcDir</code> | Folder where the generated wrapper class must be placed. |

Handling Data and Data Types in OpenL Tablets

This section includes the following topics about data and data types handling in OpenL Tablets:

- [Datatype Lifecycle](#)
- [Inheritance in Datatypes](#)
- [Byte Code Generation at Runtime](#)
- [Java Files Generation](#)
- [OpenL Internals: Accessing a Datatype at Runtime and After Building an OpenL Wrapper](#)
- [Data Table](#)

Datatype Lifecycle

1. Create a Datatype table in the rules file.
At runtime, Java class is generated for each datatype (see [Byte Code Generation at Runtime](#)).
2. If the [Generating Java Classes from Datatype Tables](#) was used, the appropriate generated Java classes have to be included in classpath (see [Java Files Generation](#)).

Inheritance in Datatypes

There is a possibility to inherit one datatype from another in OpenL Tablets. The new data type that inherits from another one has access to all fields defined in the parent data type. If a child datatype contains fields that are defined in the parent datatype, warnings or errors appear (if the field is declared with different types in the child and the parent datatype).

Also constructor with all fields of the child datatype contains all fields from the parent datatype and the *toString*, *equals* and *hashCode* methods use all fields from the parent datatype.

Byte Code Generation at Runtime

At runtime, when OpenL engine instance is being built, for each datatype component Java byte code is being generated in case there are no previously generated Java files on classpath (see [Java Files Generation](#)), it represents simple Java bean for this datatype. This byte code is being loaded to classloader so the object of type `Class<?>` can be accessed. When using this object through reflections, new instances are being created and fields of datatypes are being initialized (see the `DatatypeOpenClass` and `DatatypeOpenField` classes).

Attention! If Java class files for the datatypes on classpath have been previously generated, they will be used at runtime. It does not matter if any changes in Excel have been made. To apply these changes, remove Java files and generate Java classes from the Datatype tables (see [Generating Java Classes from Datatype Tables](#)).

Java Files Generation

As generation of datatypes is performed at runtime and developers cannot access these classes in their code, [Generating Java Classes from Datatype Tables](#) mechanism is introduced which provides the possibility of generating Java files and putting them on the file system so users can use these data types in their code.

OpenL Internals: Accessing a Datatype at Runtime and After Building an OpenL Wrapper

After parsing, each data type is put to compilation context and becomes accessible for rules during binding. All data types are placed to `IOpenClass` of the whole module and are accessible from `CompiledOpenClass#getTypes` when the OpenL wrapper is generated.

Each `TableSyntaxNode` that is of the `xls.datatype` type contains an object of data type as its member.

Data Table

A **data table** contains relational data that can be referenced as follows:

- from other tables within OpenL Tablets
- from Java code through wrappers as Java arrays
- through the OpenL Tablets run-time API as a field of the `Rules` class instance

| Data int numbers |
|------------------|
| this |
| Numbers |
| 10 |
| 20 |
| 30 |
| 40 |
| 50 |

Figure 6: Simple data table

In the example above, information in the data table can be accessed from Java code as shown in the following code example:

```
int[] num = tableWrapper.getNumbers();

for (int i = 0; i < num.length; i++) {
    System.out.println(num[i]);
}
```

where `tableWrapper` is an instance of the wrapper class of the Excel file.

| Datatype Person | |
|-----------------|------|
| String | name |
| String | ssn |

| Data Person p1 | |
|----------------|-------------|
| name | ssn |
| Name | SSN |
| Jonh | 555-55-0001 |
| Paul | 555-55-0002 |
| Peter | 555-55-0003 |
| Mary | 555-55-0004 |

Figure 7: Datatype table and a corresponding data table

In Java code, the data table `p1` can be accessed as follows:

```
Person[] persArr = tableWrapper.getP1();

for (int i = 0; i < persArr.length; i++) {
    System.out.println(persArr[i].getName() + ' ' + persArr[i].getSsn());
}
```

where `tableWrapper` is an instance of the Excel file wrapper.

3.5 Customizing Table Properties

The OpenL Tablets design allows customizing the available Table properties. The Engine employs itself to provide support of properties customization. The `TablePropertiesDefinitions.xlsx` file contains all declaration required to handle and process Table properties.

Updating table properties requires recompiling the OpenL Tablets product. The developer has to contact the OpenL Tablets provider to retrieve the table properties file. When the changes are made, the developer has to send the file back to the provider, and a new OpenL Tablets package is delivered to the developer.

Alternatively, the developer can recompile OpenL Tablets from sources of their own.

Dispatching Table Properties

Previously selecting tables that correspond to the current runtime context were processed by Java code. Now the rules dispatching is the responsibility of the generated Dispatcher Decision table (such a table will be generated for each group of methods overloaded by dimension properties). The Dispatcher table works like all decision tables so the first rule matched by properties will be executed even if there are several tables matched by properties (previously in Java code dispatching we would had got an `AmbiguousMethodException` in such a case). So, to support both functionalities, we present the system property **dispatching.mode** that has two possible values:

- **java**: dispatching is processed by Java code. The benefit of such approach is stricter dispatching: if several tables are matched by properties, the `AmbiguousMethodException` is thrown.
- **dt**: dispatching is processed by the Dispatcher Decision Table. The main benefit of this approach is performance: decision table is invoked much faster than Java code dispatching performed.

The Java approach is used by default (if the system property is not specified) or if the **dispatching.mode** property has an incorrect value.

Tables Priority Rules

To make tables dispatching more flexible, *DataTable tablesPriorityRules* in *TablePropertiesDefinitions.xlsx* is used. Each element of this table defines one rule of how to compare two tables using their properties to find more suitable table if several tables were matched by properties. Priority rules are used sequentially in comparison of two tables: if one priority rule gives result of the same priority of tables, then the next priority rule is used.

Priority rules are used differently in the Dispatcher table approach and Java code dispatching but have the same sense: select suitable table if there are several tables matched by dimension Properties.

In case of the Dispatching table, priority rules are used to sort methods of an overloaded group. Each row of the Dispatcher table represents a rule, so after sorting high-priority rules are at the top of decision tables, and in case several rows of the Decision table are fired, only the first one (of the highest priority) is executed.

In case of Java code, dispatching priority rules is used after selecting tables that correspond to the current runtime context: all matched tables are sorted in order to select one with the highest priority. If it is impossible to find the priority with the highest rule (several tables have the same priority and are of a higher priority than all other tables), *AmbiguousMethodException* is thrown.

There are two predefined priority rules and possibility to implement Java class that compares two tables using their properties:

- **min(<property name>)** – a table that has lower value of property specified will have a higher priority.
Restrictions: the property specified by name must be **instanceof Comparable<class of property value>**.
- **max(<property name>)** – a table that has a higher value of property specified will have a higher priority.
Restrictions: the same as in **min(<property name>)**.

To specify the Java comparator of tables, the following expression must be used: **javaclass:<java class name>**.
Restrictions: Java class must implement **Comparator<ITableProperties>**.

3.6 Validation for Tables

The Validation phase follows after the binding phase that serves to check all tables for errors and accumulate all errors.

Validators

All possible validators are stored in *ICompileContext* of the OpenL class. (The default compile context is *org.openl.xls.RulesCompileContext* that is generated automatically.)

Validators get the OpenL and array of *TableSyntaxNodes* that represent tables for check and must return *ValidationResult*. Validation results are as follows:

- status (fail or success) and
- all error/warning messages that have occurred

This section includes the following topics:

- [Table Properties Validators](#)
- [Existing Validators](#)

Table Properties Validators

The table properties that are described in *TablePropertyDefinition.xlsx* can have constraints. Some constraints have predefined validators associated with them.

Adding Own Property Validator

Proceed as follows:

1. Add constraint:
 1. Define constraint in `TablePropertyDefinition.xlsx` (constraints field).
 2. Create constraint class and add it into the `ConstraintFactory`.
2. Create own validator:
 1. Create class of the validator and define in method `org.openl.codegen.tools.type.TablePropertyValidatorsWrapper.init()` constraint associated with the validator.
 2. If needed, the velocity script `RulesCompileContext-validators.vm` can be modified in project `org.openl.rules.gen` that will generate `org.openl.xls.RulesCompileContext`.
 3. Run `org.openl.codegen.tools.GenRulesCode.main(String[])` to generate new `org.openl.xls.RulesCompileContext` with the validator.
3. Write unit tests!

Existing Validators

The existing validators are as follows:

- **Unique in module** validator checks uniqueness in module of a property.
- **Active table** validator checks correctness of an "active" property. Only per one active table.

3.7 Module Dependencies

Classloaders

The dependency class (datatype) resolution mechanism is implemented using specialized classloading. Each dependency has its own Java classloader so all classes used in compiling a specified module (including generated datatype Java classes) are stored in the dependency classloader.

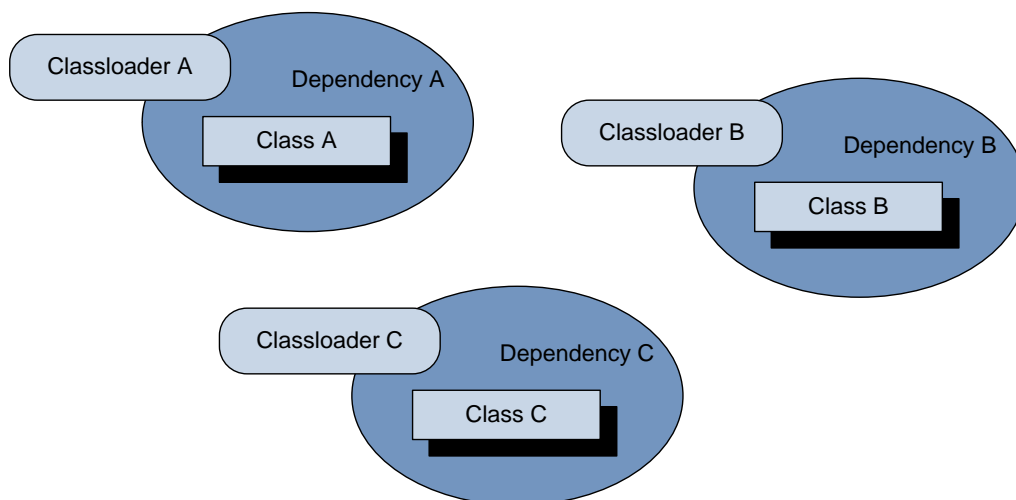


Figure 8: Dependency classloaders

The root module contains links to all of its dependencies (bundles) classloaders. When loading any class, the root module first checks if any of the bundles classloaders contains an already loaded class as shown further in Figure 9. The algorithm is as follows:

1. Get all bundle classloaders.
2. In each bundle classloader, try to find the previously loaded class.
3. If a class exists, return it. If a class does not exist, try to load the class by its classloader.
4. The class that can be found by any bundle classloader is returned.

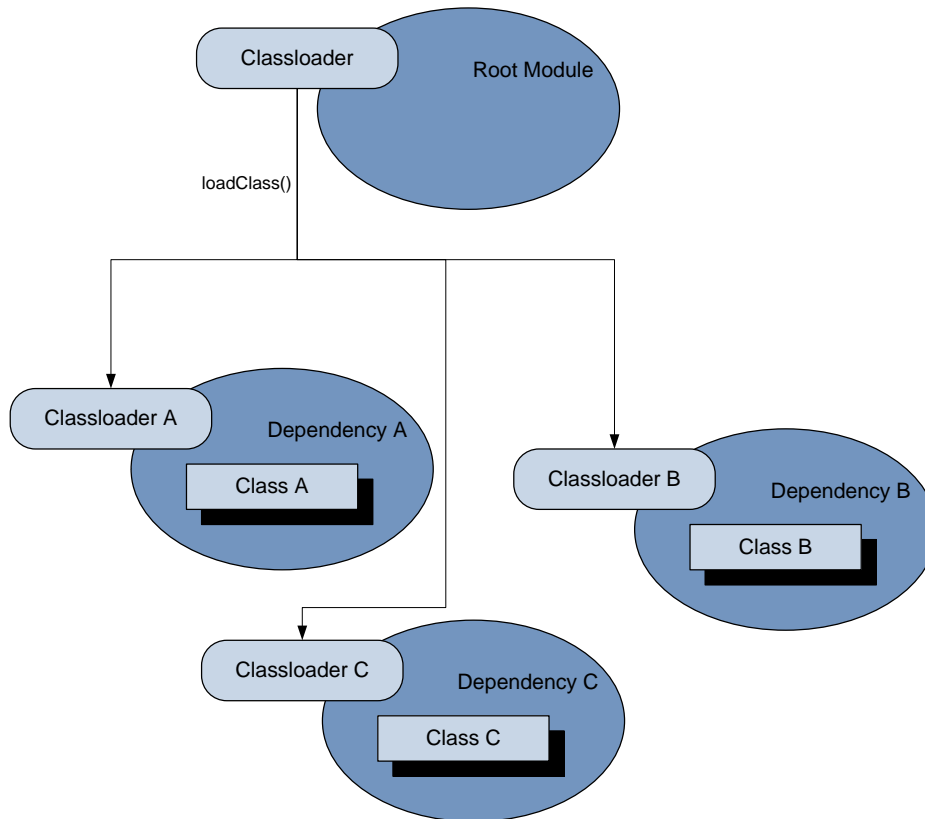


Figure 9: Load class from root module

For the dependency management feature, provide an appropriate DependencyManager object to the entry point for the OpenL compilation.

Note: Using the same class in two classloaders can cause an error because the class will be loaded by two different classloaders.

3.8 Peculiarities of OpenL Tablets Implementation

This section includes the following topics:

- [Lookup Tables Implementation Details](#)
- [Range Types Instantiation](#)

Lookup Tables Implementation Details

At first, a lookup table goes through parsing and validation. In parsing, all parts of the table, such as header, column headers, vertical conditions, horizontal conditions, return column and their values are extracted. In validation, OpenL checks if the table structure is proper.

To work with this kind of a table, the `TransformedGridTable` object is created with the constructor parameters it had in the original grid table of the lookup table (without header) and the `CoordinatesTransformer` that converts table coordinates to work with both vertical and horizontal conditions.

As a result, we get a `GridTable` and it works as a decision table structure. All coordinate transformations with lookup structure go inside. The work with columns/rows is based on the physical (not logical) structure of the table.

Range Types Instantiation

There are three ways to create `IntRange`:

1. `new IntRange(int min_number, int max_number)`: it covers all the numbers between `min_number` and `max_number`, including borders.
2. `new IntRange(Integer value)`: it covers only a given value as the beginning and the end of the range.
3. `new IntRange(String rangeExpression)`. Borders are parsed by formats of `rangeExpression`.

The same formats and restrictions are used in `DoubleRange`.

Index

E

Execution mode, 7

G

guide

audience, 4

related information, 4

typographic conventions, 4

O

OpenL Tablets

advantages, 6

basic concepts, 6

definition, 6

introduction, 6

project, 7

rules, 7

tables, 7

wrapper, 7

OpenL Tablets, 10

OpenL Tablets project

definition, 7

P

project

creating, 14

definition, 7

R

rule

definition, 7

S

system overview, 8

W

wrapper

definition, 7

generating, 15