OpenL Tablets Developer Guide

OpenL Tablets 5.11

OpenL Tablets BRMS

Document number: TP_OpenL_Dev_2.4_LSh
Revised: 09-10-2013

Table of Contents

Preface

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

The following topics are included in this preface:

- Audience
- Related Information
- Typographic Conventions

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, Eclipse, Ant, and Excel® is required to use this guide effectively.

Related Information

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

Related information	
Title	**Description**
OpenL Tablets WebStudio User's Guide	Document describing OpenL Tablets WebStudio, a web application for managing OpenL Tablets projects through web browser.
http://openl-tablets.sourceforge.net/	OpenL Tablets open source project website.

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. • 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.

Typographic styles and conventions	
Convention	**Description**
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. • Represents documentation titles.
< >	Represents placeholder values to be substituted with user specific values.
Hyperlink	Represents a hyperlink. Clicking a hyperlink displays the information topic or external source.

Chapter 1: Introducing OpenL Tablets

This section introduces OpenL Tablets and describes its main concepts.

The following topics are included in this section:

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 Java programs.

OpenL Tablets is built using the OpenL technology providing a framework for development of different language configurations.

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 Java developers.

  For example, decision tables are transformed into Java methods, and data tables become accessible as Java data arrays through the familiar getter and setter JavaBeans mechanism. 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 provides cross-indexing and search capabilities within all project documents.

OpenL Tablets supports the `.xls` file formats.

Basic Concepts

This section describes the following main OpenL Tablets concepts:

- Rules
- Tables
- Projects
- Wrappers

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 OpenLTablets 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, Java code, and Ant task for generating wrappers of table files. For detailed information on projects, see OpenLTablets 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.

Wrappers

A **wrapper** is a Java class that exposes decision tables as Java methods, data tables as Java objects and allows developers to access table information from code. To access a particular table from Java code, a wrapper Java class must be generated for the Excel file where the table is defined. 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.

Wrappers can be dynamic or static as described in the following table:

Wrapper types	
Type	**Description**
Dynamic	For a dynamic wrapper, only a rule interface must be defined upon project creation. The rules run-time factory provides instances implementing this interface in run-time. Using dynamic wrappers, and not the static wrappers, is recommended.
Static	The wrapper Java class is generated in a rule project for a static wrapper, which contains all OpenL Tablets API usage logic to call rules.

Using dynamic wrappers, rather than static wrappers, is more advantageous as it enables OpenL Tablets users to clearly define the rules displayed in the application. Using a static wrapper can be inconvenient in that a wrapper must be regenerated each time a new version of OpenL Tablets is released.

OpenL Tablets provides a specific Ant task that can be used for static generation of a wrapper from any Excel file automatically.

A static wrapper class must be regenerated in the following situations:

- A table signature, such as method name, input parameters, and return values, is modified.
- A table is added or deleted in the corresponding file.

Wrapper classes do not have to be regenerated if table data is modified or if conditions and actions are added or removed.

For more information on generating wrappers, see OpenLTablets Reference Guide, section *Generating a Wrapper*.

Execution mode for OpenL project

Execution mode for OpenL project is a light weight compilation mode that makes open class of project suitable only for evaluating of rules; but editing, tracing and search are not available. Since the Engine will not load test tables and keep in memory debug information in this mode, memory consumption is up to 5 times less than for debug mode. All rules sources are cleared after compilation and memory usage is noticeably lower than in usual mode.

By default the execution mode is used in RuleServices.

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 you use OpenL high level API (instantiation strategies) you can define an execution mode in constructor of the particular instantiation strategy
- In case you use low level API (Engine factories) you can set execution mode flag using `setExecutionMode(boolean)` method
- If you use a Static wrapper (deprecated approach!) you can use a constructor with the boolean flag "`execution mode`"

System Overview

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

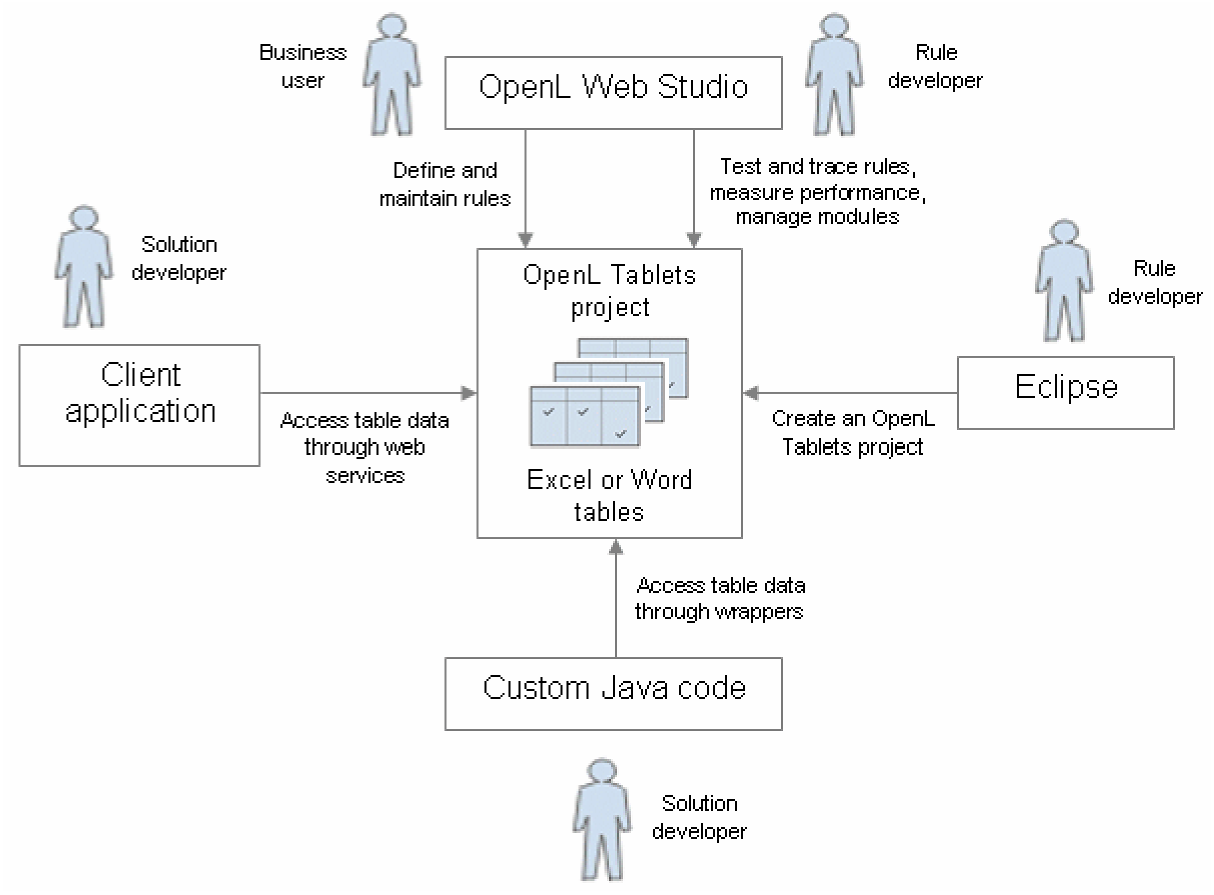

Business user
OpenL Web Studio
Rule developer
Define and maintain rules
Test and trace rules, measure performance, manage modules
Solution developer
OpenL Tablets project
Rule developer
Client application
Access table data through web services
Excel or Word tables
Eclipse
Create an OpenL Tablets project
Access table data through wrappers
Custom Java code
Solution developer

Figure 1: OpenL Tablets overview

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

1. A rule developer creates a new OpenL Tablets project in Eclipse.
2. In addition to the project itself, the rule developer also creates correctly structured tables in Excel files based on requirements and includes them in the project.
3. A business user accesses tables in the OpenL Tablets project and defines rules.

   Typically, this task is performed through OpenL Tablets WebStudio in a web browser.
4. A rule developer performs unit tests and performance tests on rules through advanced OpenL Tablets WebStudio features.
5. A developer who creates other parts of the solution employs business rules directly through the OpenL Tablets engine or remotely through web services.
6. Whenever required, the business user updates or adds new rules to project tables.

Installing OpenL Tablets

OpenL Tablets development environment is installed as an Eclipse feature delivered as part of the Exigen Process Backbone installation package. The installation process of the OpenL Tablets feature is the same as for any other Eclipse feature.

The development environment is required only for creating OpenL Tablets projects and launching OpenL Tablets WebStudio. If ready OpenL Tablets projects are accessed through OpenL Tablets WebStudio or web services, no specific software needs to be installed.

Tutorials and Examples

The OpenL Tablets Eclipse feature contains several preconfigured projects intended for new users who want to learn working with OpenL Tablets quickly.

These projects are organized into following groups:

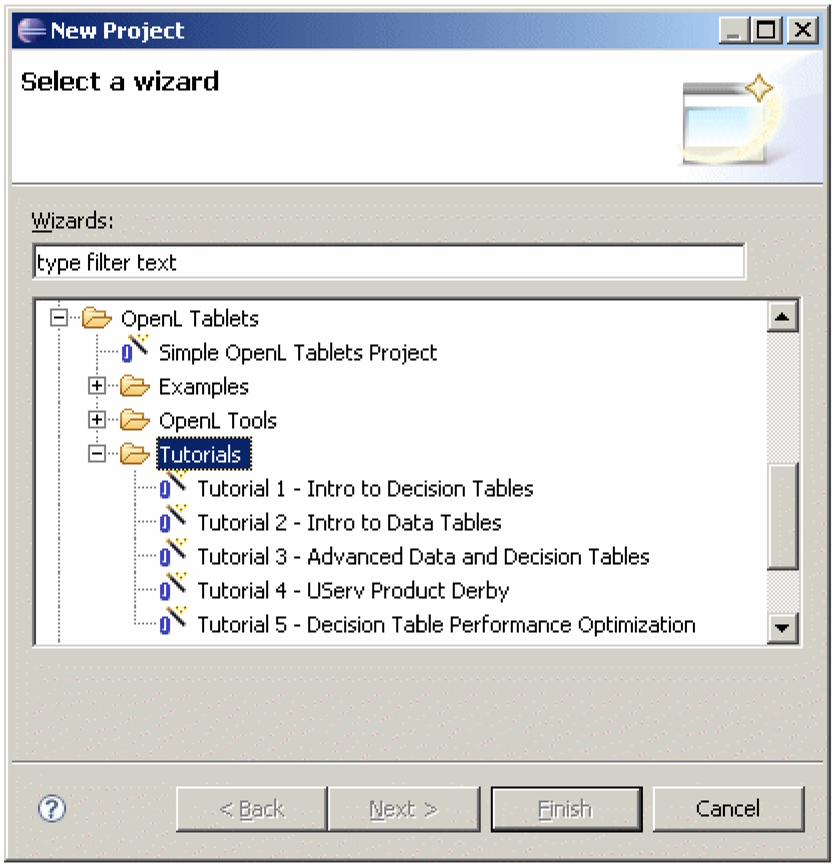
- Tutorials
- Examples

Tutorials

OpenL Tablets provides a number of tutorial projects demonstrating basic OpenL Tablets features beginning very simply and moving on to more advanced projects. Files in the tutorial projects contain detailed comments allowing new users to grasp basic concepts quickly.

To create a tutorial project, proceed as follows:

1. In Eclipse, select **File > New > Project.**
2. In the new project wizard, expand the OpenL Tablets > Tutorials folder.



OpenL Tablets
Simple OpenL Tablets Project
Examples
OpenL Tools
Tutorials
Tutorial 1 - Intro to Decision Tables
Tutorial 2 - Intro to Data Tables
Tutorial 3 - Advanced Data and Decision Tables
Tutorial 4 - UServ Product Derby
Tutorial 5 - Decision Table Performance Optimization

Figure 2: Creating tutorial projects

3. Select an appropriate tutorial project, and click **Next.**
4. In the next page, click **Finish.**

Examples

OpenL Tablets provides example projects that demonstrate how OpenL Tablets can be used in various business domains.

To create an example project, proceed as follows:

1. In Eclipse, select **File > New > Project.**
2. In the new project wizard, expand the **OpenL Tablets > Examples** folder.

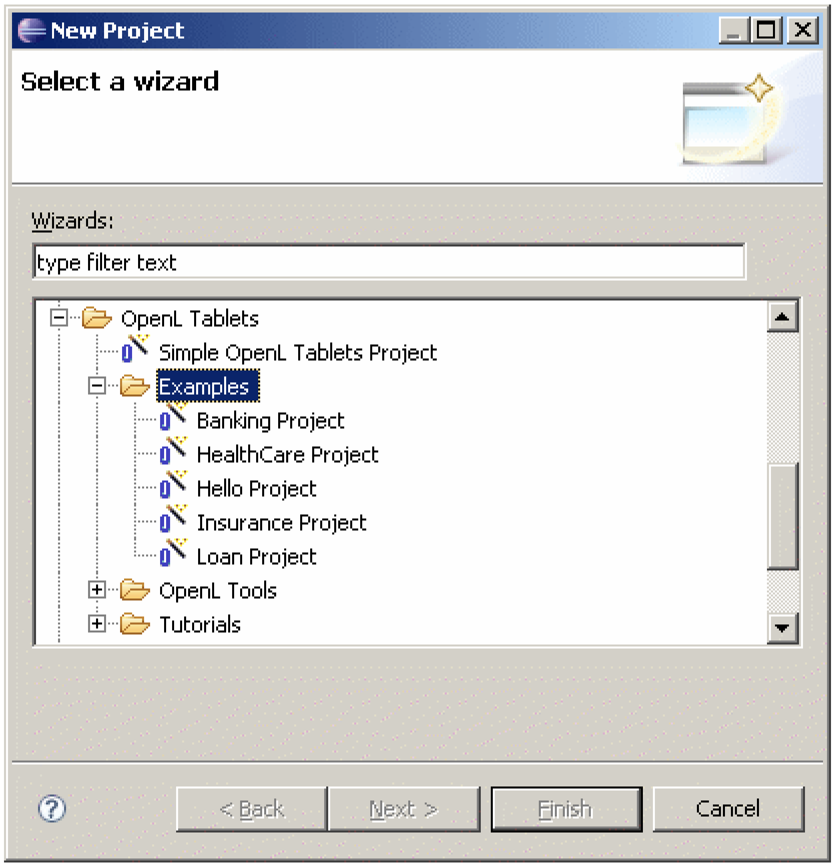


Figure 3: Creating example projects

3. Select an appropriate example project and click **Next.**
4. In the next page, click **Finish.**

Chapter 2: OpenL Tablets Rules Projects

This section describes about OpenL Tablets rules projects: how to create and how to use them.

The following topics are included in this section:

- Rules project descriptor
- Project resolving
- How to start with OpenL Rules project
- Customizing Table Properties

Rules project descriptor

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

Quick Overview

The snippet bellow is the example of rules project descriptor:

```
<project>
      <!--  Project identifier. String value which defines project uniquely. -->
      <id>my-project-id</id>
      <!-- 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>static</type>
                  <classname>com.test.MyWrapper</classname>

                  <!--
                        Rules root document. Usually excel file on file system.
                  -->
                  <rules-root path="MyModule1.xls"/>

            </module>

            <module>
                  <name>MyModule2</name>
                  <type>api</type>

                  <!--
                        Rules root document. Usually excel file on file system.
                  -->
                  <rules-root path="MyModule2.xls"/>

            </module>

      </modules>

      <!-- Project's classpath (list of all dependencies). -->
      <classpath>
            <entry path="path1"/>
            <entry path="path2"/>
      </classpath>

</project>
```

Descriptor Elements

Descriptor file contains several sections that describe projects configuration.

Project configurations

Project section		
Tag	**Required**	**Description**
id	yes	Project id. String value which defines project uniquely.
name	yes	Project name. String value which defines user-friendly project name.
comment	no	Comment for project.
modules	yes	Defines modules of project. Project can have one or more modules.
classpath	no	Project relative classpath.

Module configurations

Module section		
Tag	**Required**	**Description**
name	yes	Module name. String value which defines user-friendly module name. *Notice*: Used by WebStudio application as module display name.
type	yes	Module instantiation type. Possible values (case-insensitive): **static**, **dynamic**, **api**. Defines the way of OpenL project instantiation.
classname	yes/no	Used together with *type*. Defines name of wrapper class. Not required for *api* type.
rules-root	yes/no	Used together with *type*. Defines path to the main file of rules module.

Classpath configurations

Classpath section		
Tag	**Required**	**Description**
entry	no	Defines path to the classpath entry (classes or jar file).

Project resolving

There is a *RulesProjectResolver* that resolves all OpenL projects inside the workspace. The Resolver just lists all folders in workspace and tries to detect OpenL project by some predefined strategy. The most easy way to init *RulesProjectResolver* is to use the static method *loadProjectResolverFromClassPath() that will use **project-resolver-beans.xml** from classpath (this is usual Spring beans config that defines all resolving strategies and their order).*

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

Project Descriptor resolving strategy

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

Eclipse based resolving strategy

Previously default resolving strategy that checks that it contains "openlbuilder*"* section in ".project" file (eclipse project description file) and search all wrappers that will represent module.

By default wrappers will be searched in *gen* folder among the java classes with ending "Wrapper.java". But you can change this defaults by setting system properties:

- **org.openl.rules.wrapper.dir** – a list of directories to search for wrappers separated by ", "
- **org.openl.rules.wrapper.suffixes** – a list of wrapper endings separated by ", "

Excel file resolving strategy

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

How to start with OpenL Rules Project

OpenL Rules project as any other projects can be used as module of other project. Let's look how you can add to your project OpenL Rules project.

At first, you need to create OpenL Rules project. It can be done in following ways: using maven archetype, using eclipse plugin and manually.

Create project using Maven archetype

OpenL Tablets has archetype which can be used to create simple OpenL Rules project.

Execute in command line the following command:

```
mvn archetype:generate
```

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

After all steps are done you should have new maven based project on file system. It's a OpenL Rules project which has one module with simple rules.

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

```
mvn install
```

Create project manually

OpenL doesn't oblige user to use predefined ways of project creation process and provides way to use your own project structure. You can use OpenL Project Resolving mechanism as a base for your project structure definition. Depends on resolving strategy you need create more or less files and folders but several project's elements must be defined. See Project's elements section of the current document for more details.

Edit rules

After you created a project you need to create business rules. You can do it using OpenL Tablets WebStudio application or manually using MS Excel. If you use the simple rules project there are several simple predefined rules you have as example.

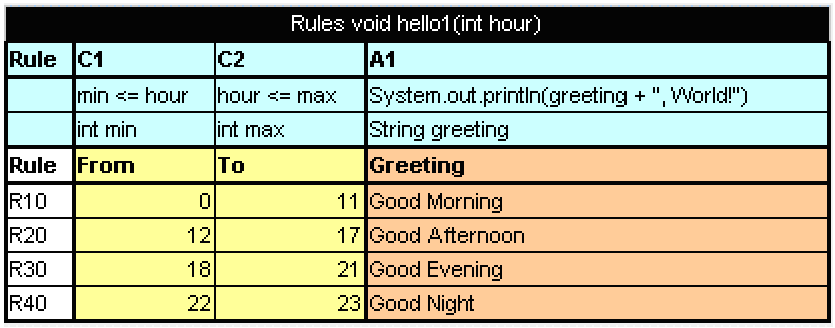
Generating a Wrapper

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

Generating a Dynamic Wrapper

Only an interface must be defined when creating a project for a dynamic wrapper. OpenL Tablets users can clearly define rules displayed in the application by using this interface. Using dynamic wrappers is a recommended practice.

This section illustrates the creation of a dynamic wrapper for a **Simple** project in Eclipse with the OpenL Tablets Eclipse Update Site installed. Only one rule **hello1** is created 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 4: 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 class for a wrapper as follows:

```
package template;

import static java.lang.System.out;
import org.openl.rules.runtime.RuleEngineFactory;

public class Dynamic_wrapper {

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

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

   }
}
```

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

Generating a Static Wrapper

To generate a static wrapper class, proceed as follows:

1. Configure the Ant task file as described in Configuring the Ant Task File.
2. Execute the Ant task file as described in Executing the Ant Task File.

For an example of how to use a static and dynamic wrapper, see Example of Using Static and Dynamic Wrappers.

Configuring the Ant Task File

Create Ant build file which will contain task to generate wrapper.

The following is an example of the build file (`GenerateJavaWrapper.build.xml` 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 table wrapper"
              targetSrcDir="gen"
       >
       </openlgen>

       <openlgen openlName="org.openl.xls" userHome="."
              srcFile="rules/Data.xls"
              targetClass=" com.exigen.claims.DataWrapper"
              displayName="Data table wrapper"
              targetSrcDir="gen"
       >
       </openlgen>

</target>
</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 `<openlgen>` section parameters:

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

Executing the Ant Task File

To execute the Ant task file and generate wrappers, proceed as follows:

1. In Eclipse, refresh the project.
2. Execute the Ant task XML file as an Ant build.
3. Refresh the project again so that wrapper classes are displayed in Eclipse.

Once wrappers are generated, the corresponding Excel files can be used in the solution.

Draw your attention that wrapper classes are generated before your project will be compiled. It means that if you use data types what are defined in OpenL rules file they will be generated into your project and then all classes (including generated ones) will be compiled. If you are going to use data types what are defined out of OpenL rules file they have to be compiled before you run the Ant task.

Example of Using a Static and Dynamic Wrapper

The following example illustrates the use of static and dynamic wrappers:

```
public class Tutorial1Main {

        public interface Tutorial1Rules {
                void hello1(int i);
        }

        public static void main(String[] args)
        {
          out.println("\n* OpenL Tutorial 1\n");

          out.println("Working using static wrapper...\n");

         callRulesWithStaticWrapper();

          out.println("\nWorking using dynamic wrapper...\n");

         allRulesWithDynamicWrapper();
        }

        private static void callRulesWithStaticWrapper() {
                //Get current hour
                Calendar calendar = Calendar.getInstance();
                int hour = calendar.get(Calendar.HOUR_OF_DAY);

                //Creates new instance of Java Wrapper for our lesson
                Tutorial_1Wrapper tut1 = new Tutorial_1Wrapper();

                //Step 1
                out.println("* Executing OpenL rules...\n");
                 // Call the method wrapping Decision Table "hello1"
                tut1.hello1(hour);
        }

        private static void callRulesWithDynamicWrapper(){
                // Creates new instance of OpenL Rules Factory
                RuleEngineFactory<Tutorial1Rules> rulesFactory =
                        new RuleEngineFactory<Tutorial1Rules>("rules/Tutorial_1.xls",
                Tutorial1Rules.class);

                //Creates new instance of dynamic Java Wrapper for our lesson
```

```
            Tutorial1Rules rules = rulesFactory.newInstance();

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

            out.println("* Executing OpenL rules...\n");
            rules.hello1(hour);
    }
}
```

Customizing Table Properties

OpenL Tablets design allows customizing list of 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. Contact your OpenL Tablets provider to retrieve the table properties file. When the changes are made, send the file back to the provider, and a new OpenL Tablets package will be delivered to you.

Table Properties Dispatching

Previously selecting tables that correspond to current runtime context have been processed by java code. Now the rules dispatching is the responsibility of 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 of functionalities we present system property "**dispatching.mode**" that has two possible values:

- **java**: dispatching will be processed by java code. The benefit of such approach is stricter dispatching: if several tables are matched by properties, the AmbiguousMethodException will be thrown.

- **dt**: dispatching will be processed by Dispatcher Decision Table. The main benefit of this approach is performance: decision table will be invoked much faster than java code dispatching performed.

The Java approach will be used by default (if the system property is not specified) or if that "**dispatching.mode**" property has a wrong value.

Tables priority rules

To make tables dispatching more flexible there exists DataTable tablesPriorityRules in `TablePropertiesDefinitions.xlsx`. 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 was matched by properties. Priority rules will be used sequentially in comparison of two tables: if one priority rule gives result of the same priority of tables then there will be used next priority rule.

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

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

In case of java code dispatching priority rules will be used after the selecting of tables that corresponds to the current runtime context: all matched tables will be sorted in order to select one of the most priority. If it is impossible to find the most priority rule (several tables has the same priority and are of more priority than all other tables) `AmbiguousMethodException` will be thrown.

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

min(<property name>) – table that has lower value of property specified will be of more priority. Restrictions: property specified by name should **be instanceOf Comparable<class of property value>**.

max(<property name>) – table that has higher value of property specified will be of more priority. Restrictions: the same as in **min(<property name>)**.

To specify java comparator of tables we should use such expression: **javaclass:<java class name>**. Restrictions: java class should implement Comparator<ITableProperties>.

Validation for Tables

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

Validators

All possible validators are stored in `ICompileContext` of 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`.
ValidationResult:

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

Table properties validators

Table properties that are described in `TablePropertyDefinition.xlsx` can have constraints. Some constraints have predefined validators associated with them.

Adding own property validator:

1. Add constraint:

1.1. Define constraint in TablePropertyDefinition.xlsx (constraints field)

1.2. Create constraint class and add it into the ConstraintFactory

2. Create own validator

   2.1. Create class of your validator and define in method org.openl.codegen.tools.type.TablePropertyValidatorsWrapper.init() constraint associated with validator.

   2.2. If it needed you can modify velocity script RulesCompileContext-validators.vm in project org.openl.rules.gen that will generate org.openl.xls.RulesCompileContext.

   2.3. Run org.openl.codegen.tools.GenRulesCode.main(String[]) to generate new org.openl.xls.RulesCompileContext with your validator.

3. Write unit tests!

Existing validators

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

Module Dependencies

Classloaders

Dependency class (datatype) resolution mechanism is implemented using specialized classloading.

Each dependency has its own java classloader. So all classes used while compiling specified module (including generated datatype java classes) are stored in its classloader.

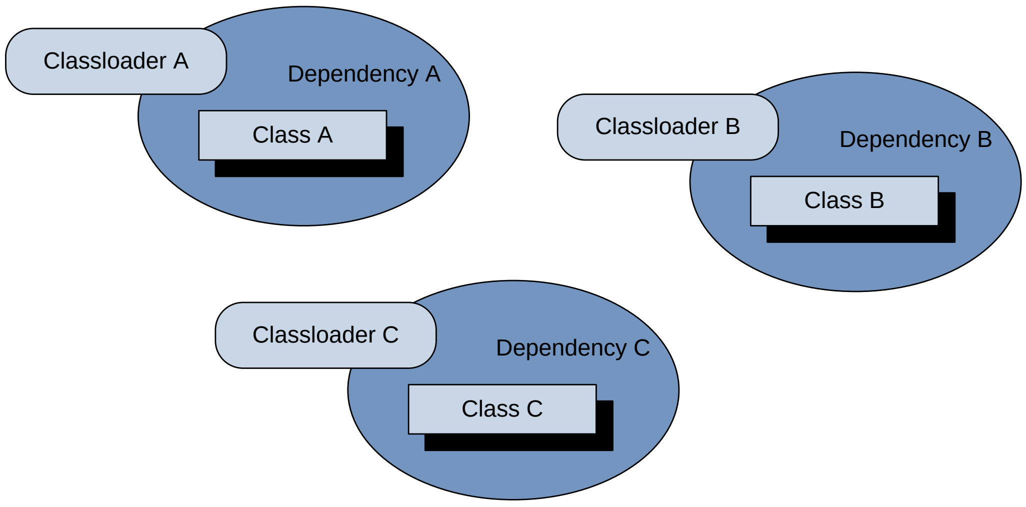


Figure 5: Dependency classloaders

The root module contains links to all his dependencies' (bundles) classloaders. When loading any class, the root module first checks his bundles classloaders if any of them contains already loaded class, as shown below. Algorithm:

1. Get all bundle classloaders

2. In each bundle classloader try to find previously loaded class.
3. If class exists, return it. If no, try to load class by this classloader.
4. Will be returned the class that can be found by any bundle classloader.

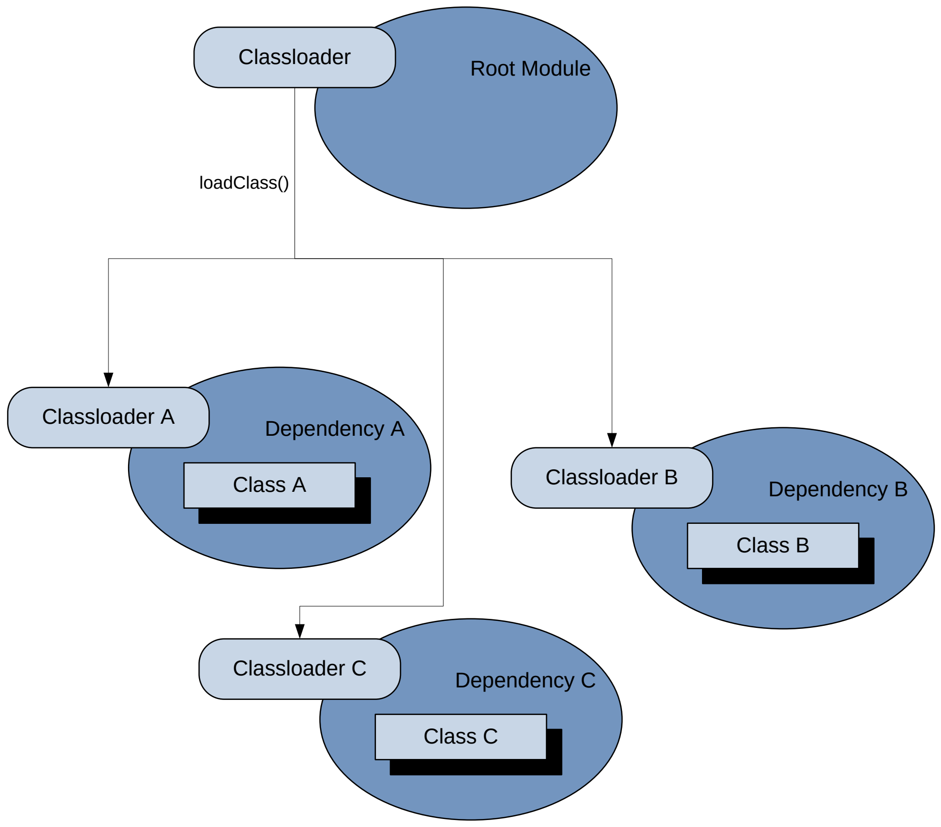


Figure 6: Load class from root module

For dependency management feature please provide appropriate DependencyManager object to the entry point for Openl compilation.

Index