



OpenL Tablets Reference Guide

Exigen Process Backbone® 7.1.0

Exigen® Decision Services

History of Revisions

History of revisions			
Revision date	Version	Author	Description
Jul 28, 2008	0.1	Juris Olekss	First draft.
Jul 30, 2008	0.2	Juris Olekss	Feedback from Siarhei Zyranau and Stanislav Shor implemented.
Aug 13, 2008	0.3	Juris Olekss	General Exigen Decision Services overview added.
Sep 8, 2008	0.4	Juris Olekss	Overview of the BEX language added.
Sep 26, 2008	1.0	Juris Olekss	Copy-editing implemented.
Dec 22, 2008	2.0	Lada Gusarova	Keywords updated.
Jan 26, 2009	2.1	Jelena Neja	Index created. EPB component names updated.

Document number: TP_EPB_7.1.0_OpenL_Ref_2.1_LG

Revised: 01-26-2009

EXIGEN CONFIDENTIAL – FOR AUTHORIZED USERS ONLY

Important Notice

Information in this document, as well as the software products and programs described in it, is furnished by Exigen Properties, Inc. and/or affiliates (Exigen) under license and may be used or copied only in accordance with the terms of such license. This document, the information contained in it, as well as the software products and programs described herein are considered confidential, proprietary, trade secrets of Exigen. You may not copy, create derivatives, decipher, decompile, develop, or otherwise reverse engineer this document, the information contained in it, or the software products and programs described. Exigen and its licensors retain all intellectual property and ownership rights to the information contained herein, as well as the software products and programs (including, but not limited to, software libraries, interfaces, source codes, documentation and training materials) described herein.

The content of this document is furnished for informational use only, is subject to change without notice, may contain technical inaccuracies or typographical errors, and should not be construed as a representation, warranty or commitment by Exigen or any other person or entity. Exigen may make improvements and/or changes in the software products and programs described in this document at any time without notice. Exigen is not responsible or liable for any changes made to this document. In particular, modifications in or to the data model described herein may have occurred or may occur in a new product release after publication of this documentation. In accordance with Exigen standard support policy, any issues arising as a result of such modifications shall not be considered support issues and shall not be covered by Exigen maintenance and support. To be clear, Exigen is providing this document and the information contained in it, "as is" without any representations, covenants, or warranties, including those of merchantability or fitness for a particular purpose. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; in that case, this notice may not apply.

Information published by Exigen on the Internet or World Wide Web may contain references or cross-references to Exigen software products and programs, as well as Exigen services that are not announced or available in your country. Such references do not imply that Exigen intends to announce such software products programs or such services in your country. Consult your local Exigen business contact for information regarding the Exigen software products and programs and Exigen services that may be available to you.

By using this document, you agree that, in no event will Exigen or its licensors be liable to any party for any direct, indirect, special or other consequential damages for any use of this document, the information contained herein, or the software products and programs described herein, including, without limitation, any lost profits, business interruption, loss of programs or other data on your information handling system or otherwise, even if Exigen were expressly advised of the possibility of such damages.

Trademark

Exigen, the Exigen logo, and Exigen Process Backbone are registered trademarks and trademarks of Exigen Properties, Inc. in the United States and/or other countries.

Microsoft and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks, service marks, or registered trademarks are the property of their respective companies.

Support contact information

Telephone:		Website:	http://www.exigengroup.com/
1-866-4EXIGEN	North America	Fax:	1-506-674-4014
1-506-674-4025	All other countries	Email:	support@exigengroup.com
1-800-508-192	Australia (within Australia only)		
0-800-448-898	New Zealand (within New Zealand only)		

Table of Contents

Preface	5
Audience	5
Related Information	5
Typographic Conventions.....	5
Chapter 1: Introducing OpenL Tablets	7
What Is Exigen Decision Services?.....	7
What Is OpenL Tablets?.....	7
Basic Concepts	8
Rules.....	8
Tables	8
Projects	9
Wrappers	9
System Overview	9
Installing OpenL Tablets.....	10
Tutorials and Examples.....	11
Tutorials	11
Examples	12
Chapter 2: Creating Tables for OpenL Tablets	13
Table Recognition Algorithm	13
Table Types.....	14
Decision Table	14
Data Type Table	19
Data Table	19
Test Table.....	24
Run Method Table	25
Method Table.....	25
Configuration Table	26
Chapter 3: Working With Projects.....	27
Project Structure.....	27
Creating a Project.....	27
Generating a Wrapper.....	28
Configuring the Ant Task File	28
Executing the Ant Task File	29
Appendix A: BEX Language Overview	30
Introduction to BEX	30
Keywords.....	30
Simplifying Expressions	31
Notation of Explanatory Variables	31
Uniqueness of Scope.....	31
Index.....	32

Preface

This preface is an introduction to the *Exigen Decision Services OpenL Tablets Reference 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.

To effectively use this guide, basic knowledge of Java, Eclipse, Ant, and Excel® is required.

Related Information

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

Related information	
Title	Description
<i>OpenL Web Studio User's Guide</i>	Document describing OpenL Web Studio, a web application for managing OpenL Tablets projects through web browser.
<i>Exigen Studio Designer's Guide</i>	Document describing the Exigen Process Backbone development environment.
http://ant.apache.org/	Website describing how to install and use Ant, a Java-based build tool.
http://openl-tablets.sourceforge.net/	OpenL Tablets open source project website.
<i>Exigen Decision Services BLS Developer's Guide</i>	Document describes BLS Rules, another rule processing framework included in Exigen Decision Services.

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.
<code>Courier</code>	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.
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. Since OpenL Tablets is part of Exigen Decision Services, a brief overview of Exigen Decision Services is also provided in this section.

The following topics are included in this section:

- [What Is Exigen Decision Services?](#)
- [What Is OpenL Tablets?](#)
- [Basic Concepts](#)
- [System Overview](#)
- [Installing OpenL Tablets](#)
- [Tutorials and Examples](#)

What Is Exigen Decision Services?

Exigen Decision Services is a collection of frameworks for rapidly creating, deploying, and maintaining rules-based systems in financial services, insurance, telecommunications, and other industries. Exigen Decision Services consists of the following main frameworks:

Main Exigen Decision Services frameworks	
Framework	Description
OpenL Tablets	Rules management framework based on tables presented in Excel and Word documents. For general overview of OpenL Tablets, see What Is OpenL Tablets? . Only this framework of Exigen Decision Services is described in this document.
Business Logic Services (BLS Rules)	Framework designed to work with rules describing constraints and logic associated with data object attributes. Rules attached to a JSF form provide user input validation that is centrally declared, reused across screens, and executed on both client and server sides. For detailed information on BLS Rules, see <i>Exigen Decision Services BLS Developer's Guide</i> .

What Is OpenL Tablets?

OpenL Tablets is a rules management framework based on tables presented in Excel and Word 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 and Word 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 or Word 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` and `.doc` 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 [Table Types](#).

Projects

An **OpenL Tablets project** is a container of all resources required for processing rule related information. Usually, a project contains Excel or Word files, Java code, and Ant task for generating wrappers of table files. For detailed information on projects, see [Chapter 3: Working With Projects](#).

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

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 or Word 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.

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

A 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 information on generating wrappers, see [Generating a Wrapper](#).

System Overview

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

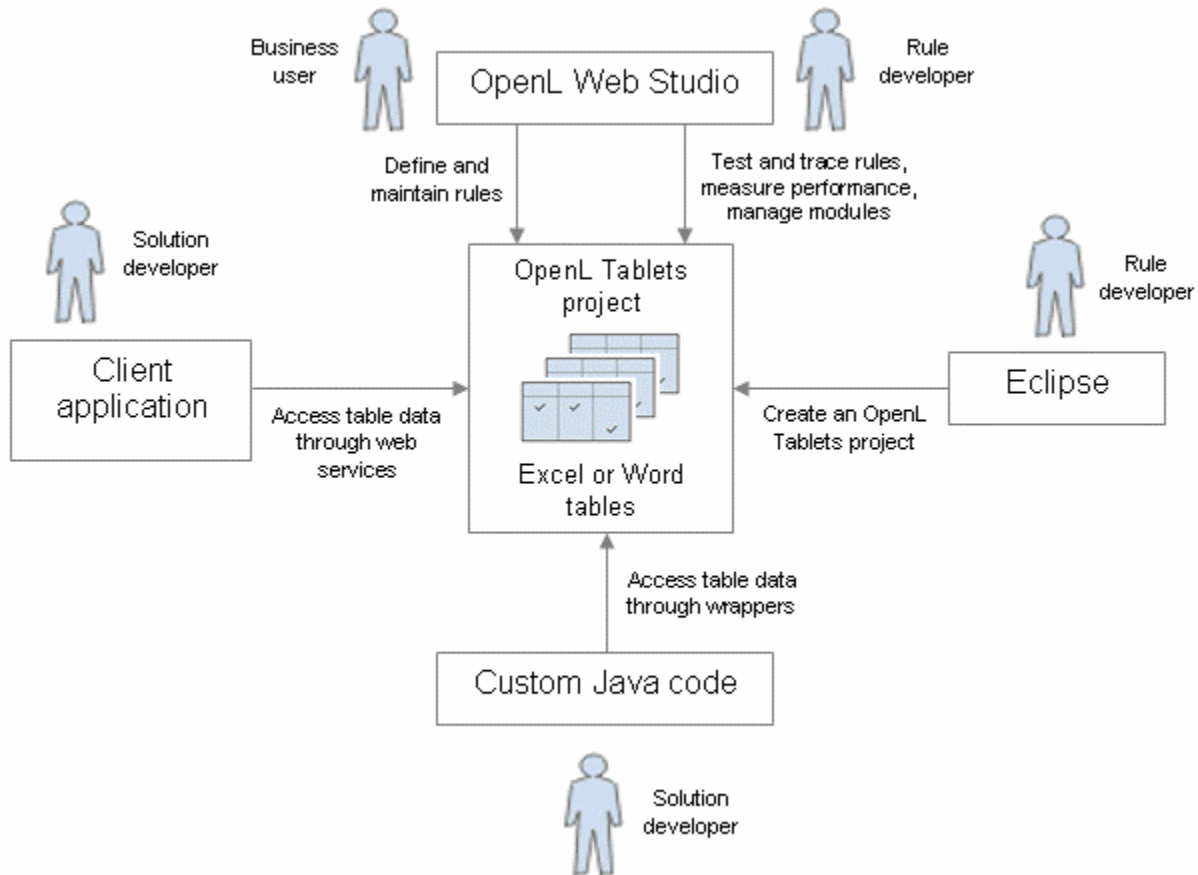


Figure 1: OpenL Tablets overview

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

1. A 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 or Word 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 Web Studio in a web browser.
4. A rule developer performs unit tests and performance tests on rules through advanced OpenL Web Studio 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 Web Studio. If ready OpenL Tablets projects are accessed through OpenL Web Studio 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 five 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.

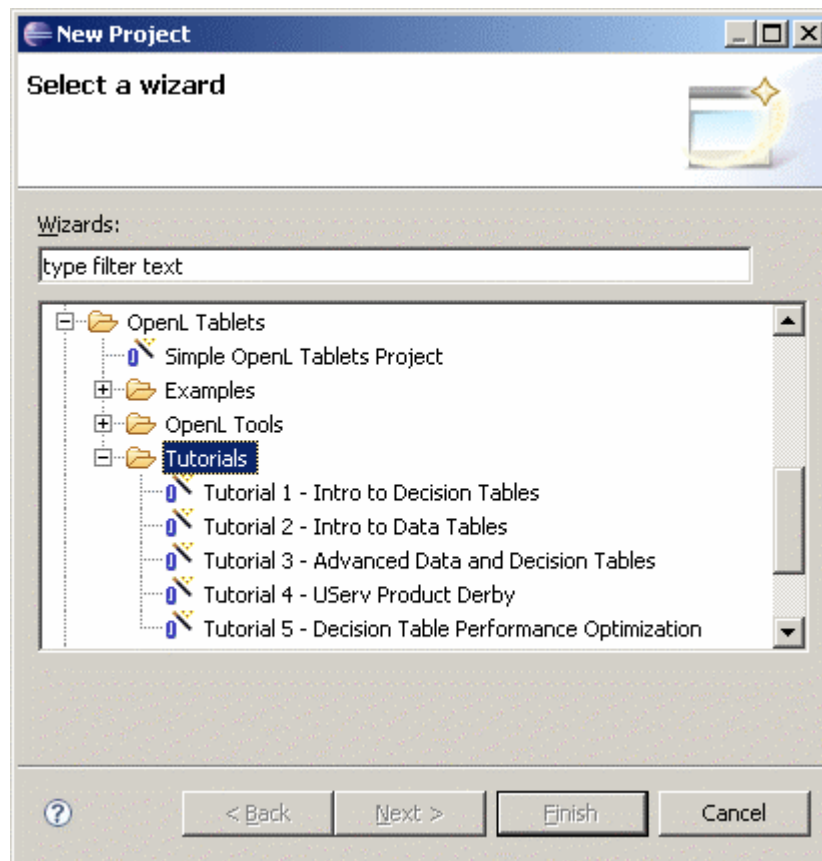


Figure 2: Creating tutorial projects

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

Examples

OpenL Tablets provides five 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.

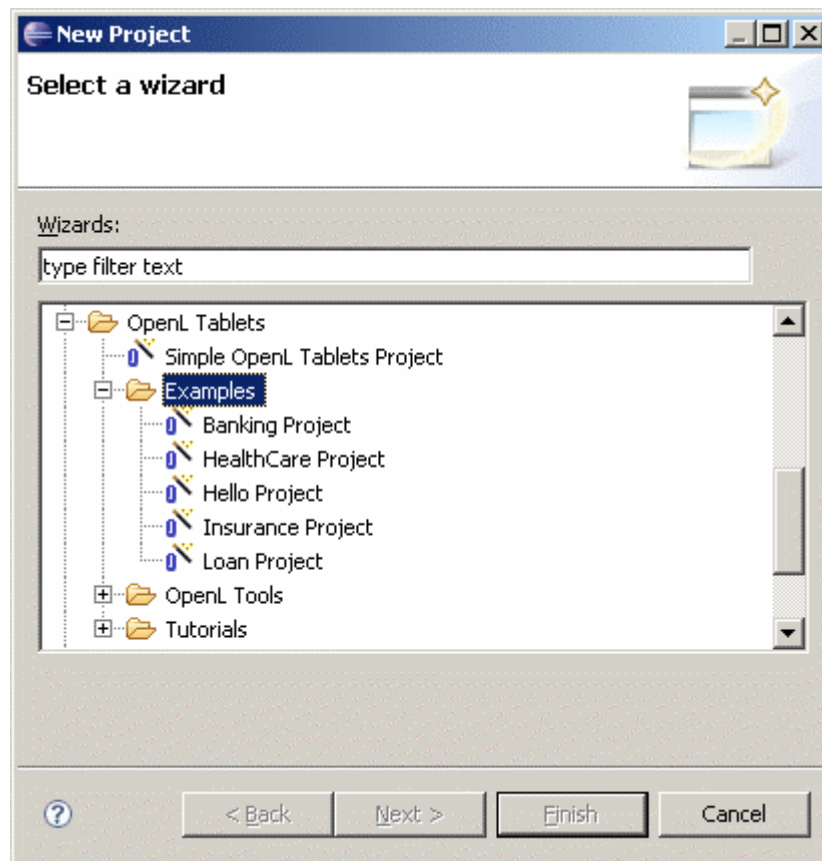


Figure 3: Creating example projects

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

Chapter 2: Creating Tables for OpenL Tablets

This section describes how OpenL Tablets processes tables and provides reference information for each table type used in OpenL Tablets.

The following topics are included in this section:

- [Table Recognition Algorithm](#)
- [Table Types](#)

Table Recognition Algorithm

This section describes the algorithm of how the OpenL Tablets engine looks for supported tables in Excel or Word files. It is important to build tables according to requirements of this algorithm; otherwise the tables will not be correctly recognized.

OpenL Tablets utilizes Excel concepts of workbooks and worksheets. These can be represented and maintained in multiple Excel files. Each workbook is comprised of one or more worksheets used to separate information by categories. Each worksheet, in turn, is comprised of one or more tables. Workbooks can include tables of different types, each of which can support a different underlying logic.

The following is the general table recognition algorithm:

1. The engine looks into each spreadsheet and tries to identify logical tables.
Logical tables must be separated by at least one empty row or column or start at the very first row or column. Table parsing is performed from left to right and from top to bottom. The first populated cell that does not belong to a previously parsed table becomes the top-left corner of a new logical table.
2. The engine reads text in the top left cell of a recognized logical table to determine its type.
If the top left cell of a table starts with a predefined keyword then such table is recognized as an OpenL Tablets table.

The following are the supported keywords:

Table type keywords	
Keyword	Table type
Rules Or DT	Decision Table
Data	Data Table
Datatype	Data Type Table
Testmethod	Test Table
Runmethod	Run Method Table
Method Or Code	Method Table
Environment	Configuration Table

All tables that do not have any of the preceding keywords in the top left cell are ignored. They can be used as comments in Excel or Word files.

3. The engine determines the width and height of the table using populated cells as clues.

It is good practice to merge all cells in the first table row, so the first row explicitly specifies the table width. The first row is called the table **header**. To put a table title before the header row, an empty row must be used between the title and the first row of the actual table.

Table Types

:table typestable:typesOpenL Tablets employs the following table types:

- [Decision Table](#)
- [Data Type Table](#)
- [Data Table](#)
- [Test Table](#)
- [Run Method Table](#)
- [Method Table](#)
- [Configuration Table](#)

Decision Table

A **decision table** contains a set of rules describing decision situations where the state of a number of conditions determines the execution of a set of actions. It is the basic table type used in OpenL Tablets decision making.

The following topics are included in this section:

- [Decision Table Structure](#)
- [Decision Table Interpretation](#)
- [Transposed Decision Tables](#)
- [Working with Arrays](#)
- [Representing Date Values](#)
- [Representing Boolean Values](#)
- [Using Calculations in Table Cells](#)

Decision Table Structure

The following is an example of a decision table:

1	Rules void hello1(int hour)		
2	properties	name	Day Hour Classification
3		category	Day and Time
4	Rule	C1	A1
5		min <= hour && hour <= max	System.out.println(greeting + ", World!")
6		int min	int max
7	Rule	From	To
8	R10	0	11
9	R20	12	17
10	R30	18	21
11	R40	22	23

Figure 4: Decision table

The following table describes its structure:

Decision table structure														
Row number	Mandatory	Description												
1	Yes	<p>Table header, which has the following pattern:</p> <pre><keyword> <method header></pre> <p>where <keyword> is either 'Rules' or 'DT' and <method header> is a signature of a method used to access the decision table and provide input parameters.</p> <p>For example, the table in the preceding diagram can be invoked as follows:</p> <pre>HelloWrapper tableWrapper = new HelloWrapper(); tableWrapper.hello1(15);</pre> <p>where <code>HelloWrapper</code> is the wrapper class of the Excel or Word file. For general information on wrappers, see Wrappers.</p> <p>Normally, this row is hidden to business users.</p>												
2 and 3	No	<p>Rows containing table properties. Each application using OpenL Tablets rules can utilize properties for different purposes.</p> <p>A predefined table property 'name' is used to specify business user oriented name of the table displayed in OpenL Web Studio.</p> <p>Although the provided decision table example contains two property rows, there can be any number of property rows in a table.</p> <p>Normally, property rows are hidden to business users.</p>												
4	Yes	<p>Row consisting of the following cell types:</p> <table> <tr> <th>Type</th><th>Description</th><th>Examples</th></tr> <tr> <td>Condition column header</td><td>Identifies that the column contains rule condition and its parameters. It must start with character 'C' followed by a number.</td><td>C1, C5, C8</td></tr> <tr> <td>Action column header</td><td>Identifies that the column contains rule actions. It must start with character 'A' followed by a number.</td><td>A1, A2, A5</td></tr> <tr> <td>Return value column header</td><td>Identifies that the column contains values to be returned to the calling program. Since there can be only one return value, the only allowed content of the cell is 'RET1'.</td><td>RET1</td></tr> </table> <p>All other cells in this row are ignored and can be used as comments.</p> <p>If a table contains action columns, the engine executes actions for all rules whose conditions are true. If a table has a return column, the engine stops processing rules after the first executed rule. If a return column has a blank cell and the rule is executed, the engine does not stop but continues checking rules in the table.</p> <p>Normally, this row is hidden to business users.</p>	Type	Description	Examples	Condition column header	Identifies that the column contains rule condition and its parameters. It must start with character 'C' followed by a number.	C1, C5, C8	Action column header	Identifies that the column contains rule actions. It must start with character 'A' followed by a number.	A1, A2, A5	Return value column header	Identifies that the column contains values to be returned to the calling program. Since there can be only one return value, the only allowed content of the cell is 'RET1'.	RET1
Type	Description	Examples												
Condition column header	Identifies that the column contains rule condition and its parameters. It must start with character 'C' followed by a number.	C1, C5, C8												
Action column header	Identifies that the column contains rule actions. It must start with character 'A' followed by a number.	A1, A2, A5												
Return value column header	Identifies that the column contains values to be returned to the calling program. Since there can be only one return value, the only allowed content of the cell is 'RET1'.	RET1												

Decision table structure										
Row number	Mandatory	Description								
5	Yes	<p>Row containing cells with code statements for condition, action, and return value column headers. OpenL Tablets supports Java grammar enhanced with OpenL Business Expression (BEX) grammar features. For information on the BEX language, see Appendix A: BEX Language Overview.</p> <p>Code in these cells can use any Java objects and methods visible to the OpenL Tablets engine. For information on enabling the OpenL Tablets engine to use custom Java packages, see Configuration Table.</p> <p>Purpose of each cell in this row depends on the cell above it as follows:</p> <table><tr><th>Cell above</th><th>Purpose</th></tr><tr><td>Condition column header</td><td><p>Specifies the logical expression of the condition. It can reference parameters in the method header and parameters in cells below.</p><p>The cell must contain an expression returning a Boolean value. All condition expressions must be true to execute a rule.</p></td></tr><tr><td>Action column header</td><td><p>Specifies code to be executed if all conditions of the rule are true. The code can reference parameters in the method header and parameters in cells below.</p></td></tr><tr><td>Return value column header</td><td><p>Specifies expression used for calculating the return value. The type of the expression must match the return value specified in the method header. This cell can reference parameters in the method header and parameters in cells below.</p></td></tr></table> <p>Normally, this row is hidden to business users.</p>	Cell above	Purpose	Condition column header	<p>Specifies the logical expression of the condition. It can reference parameters in the method header and parameters in cells below.</p> <p>The cell must contain an expression returning a Boolean value. All condition expressions must be true to execute a rule.</p>	Action column header	<p>Specifies code to be executed if all conditions of the rule are true. The code can reference parameters in the method header and parameters in cells below.</p>	Return value column header	<p>Specifies expression used for calculating the return value. The type of the expression must match the return value specified in the method header. This cell can reference parameters in the method header and parameters in cells below.</p>
Cell above	Purpose									
Condition column header	<p>Specifies the logical expression of the condition. It can reference parameters in the method header and parameters in cells below.</p> <p>The cell must contain an expression returning a Boolean value. All condition expressions must be true to execute a rule.</p>									
Action column header	<p>Specifies code to be executed if all conditions of the rule are true. The code can reference parameters in the method header and parameters in cells below.</p>									
Return value column header	<p>Specifies expression used for calculating the return value. The type of the expression must match the return value specified in the method header. This cell can reference parameters in the method header and parameters in cells below.</p>									
6	Yes	<p>Row containing parameter definition cells. Each cell in this row specifies the type and name of parameters in cells below it.</p> <p>Parameter name must be one word corresponding to Java identification rules.</p> <p>Parameter type must be one of the following:</p> <ul style="list-style-type: none">primitive Java typesJava classes visible to the engineone-dimensional arrays of the above types as described in Working with Arraysdata tables or their attributes as described in Using Advanced Data Tables <p>Normally, this row is hidden to business users.</p>								
7	Yes	<p>Descriptive column titles. The rule engine does not use them in calculations but they are intended for business users working with the table. Cells in this row can contain any arbitrary text.</p>								
8 and below	Yes	<p>Concrete parameter values.</p>								

Decision Table Interpretation

Rules inside decision tables are processed one by one in the order they are placed in the table. A rule is executed only when all its conditions are true. If at least one condition returns false, all other conditions in the same row are ignored. Absence of a parameter in a condition cell is interpreted as a true value. Blank action and return value cells are ignored.

Transposed Decision Tables

Sometimes decision tables look more convenient in transposed format where columns become rows and rows become columns. For example, a transposed version of the previously shown decision table resembles the following:

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

Figure 5: Transposed decision table

OpenL Tablets automatically detects transposed tables and is able to process them correctly.

Working with Arrays

If a parameter is an array type, there are two ways its values can be entered in a table.

The first option is to arrange array values horizontally using multiple subcolumns. The following is an example of this approach:

String[] set				
Number Set				
1	3	5	7	9
2	4	6	8	

Figure 6: Arranging array values horizontally

In this example, the contents of the `set` variable for the first rule are `[1,3,5,7,9]` and for the second rule `[2,4,6,8]`. Values are read from left to right.

The second option is to present parameter values vertically as follows:

String[] set	
#	Number Set
1	1
	3
	5
	7
	9
2	2
	4
	6
	8

Figure 7: Arranging array values vertically

In the second case, the boundaries between rules are determined by the height of the leftmost cell. Therefore, an additional column must be added to the table to specify boundaries between arrays.

In both cases, empty cells are not added to the array.

To facilitate work with arrays, OpenL Tablets provides the following method, which determines whether a particular element is included in an array:

```
contains(Object[] ary, Object obj)
```

This method works only with Java objects, not primitive Java types. The following example displays a table using the `contains` method:

Rules String getType1(String num)					
C1					RET1
contains(set, num)					result
String[] set					String result
Number Set					Result
1	3	5	7	9	Odd
2	4	6	8		Even

Figure 8: Checking arrays in conditions

Representing Date Values

To represent date values in table cells, the following format must be used:

```
'<month>/<date>/<year>
```

The value must always be preceded with an apostrophe. Excel treats these values as plain text and does not convert to any specific date format.

The following are valid date value examples:

```
'5/7/1981
```

```
'10/20/2002
```

Representing Boolean Values

OpenL Tablets supports the following formats of Boolean values:

- True, TRUE, Yes, YES
- False, FALSE, No, NO

Using Calculations in Table Cells

OpenL Tablets can perform mathematical calculations involving method input parameters in table cells. For example, instead of returning a concrete number, a rule can return a result of a calculation involving one of the input parameters. Calculation result type must match the type of the cell. Text in cells containing calculations must start with an apostrophe followed by `=`. Excel treats such values as plain text.

The following decision table demonstrates calculations in table cells:

Rules int ampmTo24(int ampmHr, String ampm)		
C1	C2	RET1
range.contains(ampmHr)	suffix.equals(ampm)	result
IntRange range	String suffix	int result
AM/PM hour	AM or PM	24 hour
12	AM	0
1-11	AM	=ampmHr
12	PM	12
1-11	PM	=ampmHr+12

Figure 9: Decision table with calculations

The table transforms a 12 hour time format into a 24 hour time format. The column `RET1` contains two cells that perform calculations with the input parameter `ampmHr`.

Calculations use regular Java syntax, similar to what is used in conditions and actions.

Note: Excel formulas are not supported by OpenL Tablets.

Data Type Table

A **data type table** defines an OpenL Tablets data carrier. Data types can be used in data tables. For information on how this is done, see [Data Table](#).

The following is an example of a data type table defining a data type called Person:

Datatype Person	
String	name
String	ssn
Date	dob
String	gender
String	maritalStatus

Figure 10: Data type table

The first row is the header containing the keyword 'Datatype' followed by the name of the data type. Every row, beginning with the second one, represents one property of the data type. The first column contains property types; the second column contains corresponding property names.

Data Table

A **data table** contains relational data that can be referenced from other tables within OpenL Tablets. In addition, information in data tables can be accessed from Java code through wrappers as Java arrays. Data tables can contain Java classes or OpenL Tablets data types. For information on data types, see [Data Type Table](#).

The following topics are included in this section:

- [Using Simple Data Tables](#)
- [Using Advanced Data Tables](#)
- [Ensuring Data Integrity](#)
- [Specifying Data for Aggregated Objects](#)

Using Simple Data Tables

The following is an example of a data table containing a simple array of numbers:

Data int numbers
this
Numbers
10
20
30
40
50

Figure 11: Simple data table

The first row is the header containing text in the following format:

```
Data <data table type> <data table name>
```

In simple data tables, the keyword 'this' must be used for the following types:

- all primitive Java types
- class `java.lang.String`
- class `java.util.Date`
- all Java classes with a public constructor with a single String parameter

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 or Word file. For information on wrappers, see [Wrappers](#).

Using Advanced Data Tables

Advanced data tables are used for storing information for complex constructions, such as Java beans and data types. For information on data types, see [Data Type Table](#).

The first row of an advanced data table contains text in the following format:

```
Data <Java bean or data type> <data table name>
```

Each cell in the second row contains an attribute name of the data type or Java bean. Normally, the second row is hidden to business users.

The third row contains attribute display names. Each row starting with the fourth one contains values for specific data type instances.

The following diagram shows a data type table and a corresponding data table with concrete values below it:

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 12: Data type table and a corresponding data table

Data tables can use Java beans instead of data types. For example, instead of using a data type table Person, a developer can use the following Java bean:

```
public class Person {

    String name;
    String ssn;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String getSsn() {
        return ssn;
    }
    public void setSsn(String ssn) {
        this.ssn = ssn;
    }

}
```

If a Java bean is used, to avoid compilation error, the package where the Java bean is located must be imported using a configuration table as described in [Configuration 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());
}
```

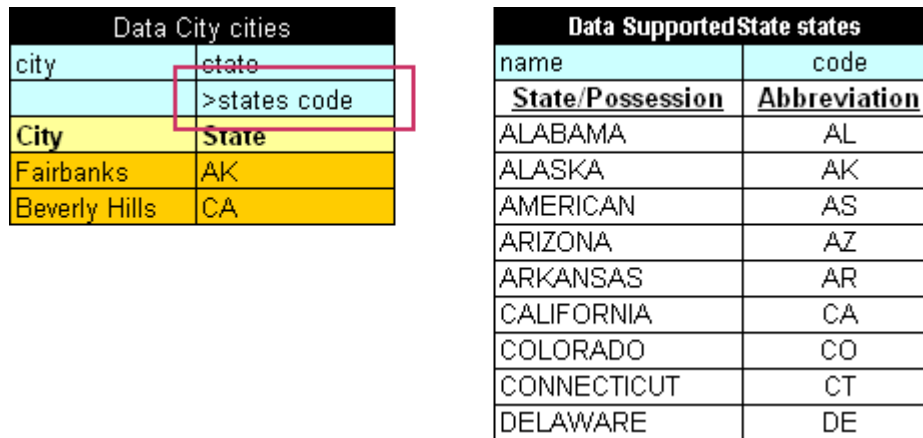
where `tableWrapper` is an instance of the Excel or Word file wrapper. For information on wrappers, see [Wrappers](#).

Ensuring Data Integrity

If a data table contains values defined in another data table, it is important to specify this relationship so that OpenL Tablets can check data integrity during compilation. The relationship between two data tables is defined using foreign keys, a concept that is used in database management systems. Reference to another data table must be specified in an additional row below the row where attribute names are entered. The following format must be used:

```
> <referenced data table name> <column name of the referenced data table>
```

In the following diagram, the data table **cities** contains values from the table **states**. To ensure users enter correct values, a reference to the **code** column in the **states** table is defined.



city	state
	>states code
City	State
Fairbanks	AK
Beverly Hills	CA

name	code
State/Possession	Abbreviation
ALABAMA	AL
ALASKA	AK
AMERICAN	AS
ARIZONA	AZ
ARKANSAS	AR
CALIFORNIA	CA
COLORADO	CO
CONNECTICUT	CT
DELAWARE	DE

Figure 13: Defining a reference to another data table

In case user enters invalid state abbreviation in the table **cities**, OpenL Tablets reports an error.

The target column does not have to be specified if it is the first column in the referenced data table. For example, if a reference was made to the column **name** in the table **states**, the following simplified reference could be used:

```
>states
```

Note: To ensure users enter correct values, cell data validation lists can be used in Excel limiting the range of values users can type in.

Specifying Data for Aggregated Objects

Data tables can be used to specify attributes of referenced objects. An object referenced by a data table is called an **aggregated object**. To specify an attribute of an aggregated object, the following format must be used in the row containing data table attribute names:

```
<name of reference to the aggregated object>.<object attribute>
```

To illustrate this approach, assume there are two Java classes `ZipCode` and `Address` defined:

```
public class ZipCode {
    String zip1; // 5-digit part - mandatory
    String zip2; // 4-digit part - optional
}
```

```

        public String getZip1() {
            return zip1;
        }
        public void setZip1(String zip1) {
            this.zip1 = zip1;
        }
        public String getZip2() {
            return zip2;
        }
        public void setZip2(String zip2) {
            this.zip2 = zip2;
        }
    }

    public class Address {
        String street;
        String city;
        ZipCode zip;

        public String getCity() {
            return city;
        }
        public void setCity(String city) {
            this.city = city;
        }
        public String getStreet() {
            return street;
        }
        public void setStreet(String street) {
            this.street = street;
        }
        public ZipCode getZip() {
            return zip;
        }
        public void setZip(ZipCode zip) {
            this.zip = zip;
        }
    }

```

As can be seen from the code, the `Address` class contains a reference to the `ZipCode` class. A data table can be created that specifies values for both classes at the same time, for example:

Data Address addresses			
street	city	zip.zip1	zip.zip2
Street1	City	Zip1	Zip2
1600 Pennsylvania Avenue	Washington	20500	
1085 Summit Dr	Beverly Hills	90210	2814

Figure 14: Specifying values for aggregated objects

In the preceding example, columns **Zip1** and **Zip2** contain values for class `ZipCode` referenced by class `Address`.

All Java classes referenced in a data table must be imported using a configuration table as described in [Configuration Table](#).

Note: The reference path can be of any arbitrary depth, for example `account.person.address.street`.

Test Table

A **test table** is used to perform unit tests on decision tables and method tables. It calls a particular method, provides test input values, and checks whether the returned value matches the expected value. Test tables are mostly used for testing decision tables.

Note: Test tables can be used to execute any Java method but in that case a method table must be used as a proxy.

For example, in the following diagram, the table on the left is a decision table but the table on the right is a unit test table that tests data of the decision table:

Rules int ampmTo24(int ampmHr, String ampm)			Testmethod ampmTo24 ampmTo24Test		
C1	C2	RET1	ampmHr	ampm	res_
range.contains	suffix.equals	result	Hour	AM/PM	24 Hr
IntRange range	String suffix	int result			
AM/PM hour	AM or PM	24 hour			
12	AM	0		3 AM	3
1-11	AM	=ampmHr		12 AM	0
12	PM	12		12 PM	12
1-11	PM	=ampmHr+12		3 PM	15

Figure 15: Decision table and its unit test table

A test table has the following structure:

- The first row is the table header, which has the following format:
Testmethod <method name> <test table name>
'Testmethod' is a keyword that identifies a test table. The second parameter is the name of the decision table method or any other Java method to be tested. The third parameter is the name of the test table, which is also the name of the method by which the test table can be executed from Java code.
- The second row provides a separate cell for each input parameter of the decision table method followed by column **_res_**, which contains the expected test result values.
- The third row contains display values intended for business users.
- Starting with the fourth row, each row is an individual test run.
When a test table is called, the OpenL Tablets engine calls the specified method for every row in the test table and passes the corresponding input parameters to it.

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:

```
TableWrapper tableWrapper = new TableWrapper();

TestResult tr = tableWrapper.insuranceTestTestAll();

System.out.println("Number of failed test runs: "+tr.getNumberOfFailures());
```


If OpenL Tablets projects are accessed and modified through OpenL Web Studio, the user interface provides more advanced and convenient utilities for running tests and viewing test results. For information on using OpenL Web Studio, see *OpenL Web Studio User's Guide*.

Run Method Table

A **run method table** calls a particular decision table or method table multiple times and provides input values for each individual call. Therefore, run method tables are similar to test tables, except they do not perform a check of values returned by the called method.

Note: Run method tables can be used to execute any Java method but in that case a method table must be used as a proxy.

The following is an example of a run method table:

Runmethod append appendRun	
firstWord	secondWord
Hi,	John!
Hello,	Mary!
Good morning,	Bob!

Figure 16: Run method table

This example assumes there is a method `append` defined with two input parameters, `firstWord` and `secondWord`. The run method table calls this method three times with three different sets of input values.

A run method table has the following structure:

- The first row is a table header, which has the following format:
`Runmethod <method to call> <run method table name>`
- The second row contains cells with method input parameter names.
- Starting with the third row, each row is a set of input parameters to be passed to the called method.

Method Table

A **method table** is a Java method described within a table. The following is an example of a method table:

Method String getGreeting(String name)
return "Hi, "+name;

Figure 17: Method table

The first row is a table header, which has the following format:

`<keyword> <return type> <method name and parameters>`

where `<keyword>` is either 'Method' or 'Code'.

The second row is the actual code to be executed. It can reference parameters passed to the method and all Java objects and tables visible to the OpenL Tablets engine.

Method in the preceding example table can be called from Java code as follows:

```
ApprovalRulesWrapper tableWrapper = new ApprovalRulesWrapper();  
  
System.out.println(tableWrapper.getGreeting("John"));
```

Configuration Table

OpenL Tablets allows externalizing business logic into Excel or Word files. However, these files can still use objects and methods defined in the Java environment. To enable use of Java objects and methods in Excel or Word tables, the file must have a configuration table. A **configuration table** provides information to the OpenL Tablets engine about available Java packages. Another purpose of a configuration file is to point to other Excel and Word files that can be referenced in tables.

A configuration table is identified by the keyword 'Environment' in the first row. No additional parameters are required. Starting with the second row, a configuration table must have two columns. The first column contains commands and the second column contains input strings for commands.

The following commands are supported in configuration tables:

Configuration table commands	
Command	Description
import	Imports the specified Java package so that its objects and methods can be used in tables.
include	Includes another Excel or Word file so that its tables and data can be referenced in tables of the current file.

The following is an example of a configuration table:

Environment	
	com.exigen.claims.data
import	org.openl.meta
include	../include/Approval_TestData.xls

Figure 18: Configuration table

Chapter 3: Working With Projects

This section describes creating an OpenL Tablets project. For general information on projects, see [Projects](#).

The following topics are included in this section:

- [Project Structure](#)
- [Creating a Project](#)
- [Generating a Wrapper](#)

Project Structure

To use the OpenL Tablets rule technology in a solution, an OpenL Tablets project must be created in Eclipse. An OpenL Tablets project is an Exigen Studio project with an OpenL Tablets facet. Typically, an OpenL Tablets project contains the following general elements:

OpenL Tablets project contents	
Element	Description
Excel or Word files	Physical storage of rules and data in the form of tables.
Ant task for generating file wrappers	Ant configuration file used for creating wrapper classes for Excel or Word files so that they can be accessed from code. For general information on wrappers, see Wrappers .
Wrappers	Java classes providing access to OpenL Tablets objects in Excel or Word files. Wrappers must be generated as described in Generating a Wrapper .
Additional Java code	Optional classes for domain models and for processing or testing rules in the project. Solution developers can decide whether to include additional code in OpenL Tablets projects.

The following table describes OpenL Tablets specific folders in the physical project structure:

OpenL Tablets project structure	
Folder	Contents
src	Contains all project Java classes apart from wrappers.
rules	Contains Excel or Word files.
gen	Contains generated wrapper classes.
bin	Contains compiled Java code.
build	Contains Ant configuration file for generating wrapper classes. For information on generating wrappers, see Generating a Wrapper .

Creating a Project

To create a new OpenL Tablets project, create an Exigen Studio project and enable the OpenL Tablets facet in the project as described in *Exigen Studio Designer's Guide*.

A new project is created containing simple template files that developers can use as the basis for custom rule solutions.

Generating a Wrapper

Access to rules and data in Excel or Word tables is realized through wrapper classes. For general information on wrappers, see [Wrappers](#).

To generate a 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](#).

Configuring the Ant Task File

When a new OpenL Tablets project is created, it already contains an Ant task file `GenerateJavaWrapper.build.xml` located in the `build` folder. When the file is executed, it automatically creates wrapper Java classes for specified Excel or Word files. The Ant task file must be adjusted to match contents of the specific project.

For each Excel or Word 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 <code><openlgen></code> section	
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 or Word file for which a wrapper class must be generated.
<code>targetClass</code>	Full name of the wrapper class to be generated. OpenL Web Studio 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 Web Studio.
<code>displayName</code>	End user oriented title of the file that appears in OpenL Web Studio.
<code>targetSrcDir</code>	Folder where the generated wrapper class must be placed.

The following is an example of the `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="." />
  </target>
</project>
```

```
        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>
```

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 or Word files can be used in the solution.

Appendix A: BEX Language Overview

This section provides a general overview of the BEX language that can be used in OpenL Tablets expressions.

The following topics are included in this section:

- [Introduction to BEX](#)
- [Keywords](#)
- [Simplifying Expressions](#)

Introduction to BEX

BEX language allows a flexible combination of grammar and semantics by extending the existing Java grammar and semantics presented in the `org.openl.j` configuration using new grammar and semantic concepts. It enables users to write expressions similar to natural human language.

BEX does not require any special mapping; the existing Java business object model automatically becomes the basis for open business vocabulary used by BEX. For example, Java expression 'policy.effectiveDate' is equivalent with BEX expression 'Effective Date of the Policy'.

If the Java model correctly reflects business vocabulary, there is no further action required. In case the Java model is not satisfactory, custom type-safe mapping or renaming can be applied.

Keywords

The following table represents BEX keyword equivalents to Java expressions:

BEX keywords	
Java expression	BEX equivalents
==	<ul style="list-style-type: none"> • equals to • same as
!=	<ul style="list-style-type: none"> • does not equal to • different from
a.b	b of the a
<	is less than
>	is more than
<=	<ul style="list-style-type: none"> • is less or equal • is in
!>	is no more than
>=	is more or equal
!<	is no less than

Because of these keywords, name clashes with business vocabulary can occur. The easiest way to avoid clashes is to use upper case notation when referring to model attributes because BEX grammar is case sensitive and all keywords are in lower case.

For example, assume there is an attribute called `isLessThanCoverageLimit`. If it is referred as 'is less than coverage limit', a name clash with keywords 'is less than' occurs. The workaround is to refer to the attribute as 'Is Less Than Coverage Limit'.

Simplifying Expressions

Unfortunately, the more complex an expression is, the less comprehensible the natural language expression becomes in BEX. For this purpose, BEX provides the following methods for simplifying expressions:

- [Notation of Explanatory Variables](#)
- [Uniqueness of Scope](#)

Notation of Explanatory Variables

BEX supports a notation where an expression is written using simple variables followed by the attributes they represent. For example, assume the following expression is used in Java:

```
(Agreed Value of the vehicle - Market Value of the vehicle) / Market Value of the vehicle is  
more than Limit Defined By User
```

As can be seen from the example, the expression is hard to read. However, the expression is much simpler if written according to the notion of explanatory variables as follows:

```
(A - M) / M > X, where  
A - Agreed Value of the vehicle,  
M - Market Value of the vehicle,  
X - Limit Defined By User
```

This syntax is similar to the one used in scientific publications and is much easier to read for complex expressions. It provides a good mix of mathematical clarity and business readability.

Uniqueness of Scope

BEX provides another way for simplifying expressions using the concept of unique scope. For example, if there is only one policy in the scope of expression, the user can write 'effective date' instead of 'effective date of the policy'. BEX automatically determines the uniqueness of the attribute and either produces a correct path or emits an error message in case of ambiguous statement. The level of the resolution can be modified programmatically and by default equals 1.

Index

A

- aggregated object
 - definition, 22
 - specifying data, 22
- ant task file
 - configuring, 28
 - executing, 29
- arrays, 17

B

- BEX language, 30
 - explanatory variables, 31
 - introduction, 30
 - keywords, 30
 - simplifying expressions, 31
 - unique scope, 31
- Boolean values
 - representing, 18

C

- calculations
 - using in table cells, 18
- configuration table, 26

D

- data integrity, 22
- data table
 - advanced, 20
 - definition, 19
 - simple, 20
- data type table
 - definition, 19
- date values
 - representing, 18
- decision table
 - definition, 14
 - interpretation, 16
 - structure, 14
 - transposed, 17

E

- examples, 11, 12
- Exigen Decision Services
 - definition, 7

G

- guide
 - audience, 5
 - related information, 5

- typographic conventions, 5

M

- method table
 - definition, 25

O

- OpenL Tablets
 - advantages, 8
 - basic concepts, 8
 - creating a project, 27
 - creating tables, 13
 - definition, 7
 - installing, 10
 - introduction, 7
 - project, 9
 - rules, 8
 - table types, 14
 - tables, 8
 - wrapper, 9
- OpenL Tablets project
 - definition, 9

P

- project
 - creating, 27
 - definition, 9
 - structure, 27

R

- rule
 - definition, 8
- run method table
 - definition, 25
 - structure, 25

S

- system overview, 9

T

- table
 - types, 14
- table cells
 - using calculations, 18
- table recognition
 - algorithm, 13
- test table
 - definition, 24
 - structure, 24
- tutorials, 11

W

wrapper

definition, 9
generating, 28