# Oracle® Business Rules

User's Guide

10*g* Release 3 (10.1.3)

**B15986-01**

January 2006

ORACLE®

Oracle Business Rules User's Guide 10*g* Release 3 (10.1.3)

B15986-01

Copyright © 2006, Oracle. All rights reserved.

Primary Author: Thomas Van Raalte

Contributing Author: Kevin Yu Hwang

Contributors: Qun Chen, Ching Luan Chung, David Clay, Kathryn Gruenefeldt, Gary Hallmark, Phil Varner, Neal Wyse, Lance Zaklan

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

# Contents


## 1 Overview of Oracle Business Rules

1.1 Introduction to Oracle Business Rules ........ 1-2
1.1.1 What Are Business Rules? ........ 1-2
1.1.2 What Is a Data Model? ........ 1-2
1.1.3 What Is a Rule-Based System? ........ 1-3
1.2 Oracle Business Rules Components ........ 1-4
1.2.1 Introducing Oracle Business Rules Rule Author ........ 1-4
1.2.2 Introducing Oracle Business Rules Rules SDK ........ 1-5
1.2.3 Introducing Oracle Business Rules RL Language ........ 1-5
1.2.4 Introducing Oracle Business Rules Rules Engine ........ 1-5
1.3 Oracle Business Rules Rule Author Terms and Concepts ........ 1-6
1.3.1 Working with Rules ........ 1-6
1.3.2 Working with Rule Sets ........ 1-6
1.3.3 Working with Repositories and Dictionaries ........ 1-7
1.3.4 Working with Facts ........ 1-7
1.3.5 Working with Functions Variables and Constraints ........ 1-8
1.4 Steps for Rule Enabling a Java Application ........ 1-9
1.4.1 Identify Application Areas to Rule Enable ........ 1-9
1.4.2 Provide Rule Author Definitions for the Data Model ........ 1-9
1.4.3 Develop a Business Vocabulary for the Data Model ........ 1-9
1.4.4 Write and Customize Rules ........ 1-10
1.4.5 Modify or Create Application Logic that Uses the Rules Engine ........ 1-10
1.4.6 Test the Rule Enabled Application ........ 1-10

## 2 Getting Started with Rule Author

2.1 Creating a Rule Author User ........ 2-1
2.2 Starting Rule Author ........ 2-2
2.3 Rule Author Home Page ........ 2-4
2.4 Creating and Saving a Dictionary for the Car Rental Sample ........ 2-4
2.4.1 Connecting to a Rule Author Repository ........ 2-5
2.4.2 Creating a Rule Author Dictionary ........ 2-7
2.4.3 Saving a Rule Author Dictionary with a Version ........ 2-8
2.4.4 Saving a Rule Author Dictionary ........ 2-8
2.5 Defining a Data Model for the Car Rental Sample ........ 2-9
2.5.1 Using Java Objects as Facts in the Car Rental Sample ........ 2-9

2.5.2 Adding Java Classes and Packages to Rule Author .......... 2-9
2.5.3 Importing Java Classes to a Data Model .......... 2-11
2.5.4 Saving the Current State of Definitions .......... 2-13
2.6 Defining Business Vocabulary for the Car Rental Sample .......... 2-13
2.6.1 Specifying the Business Vocabulary for Java Fact Definitions .......... 2-13
2.6.2 Specifying the Business Vocabulary for Functions .......... 2-14
2.6.3 Specifying the Visibility for Properties and Methods .......... 2-14
2.7 Defining a Rule for the Car Rental Sample .......... 2-15
2.7.1 Creating a Rule Set for the Car Rental Sample .......... 2-15
2.7.2 Creating a Rule for the Car Rental Sample .......... 2-16
2.8 Customizing Rules for the Car Rental Sample .......... 2-24
2.9 Creating a Java Application Using Oracle Business Rules .......... 2-25
2.9.1 Importing the Rules SDK and Rules RL Classes .......... 2-26
2.9.2 Initialize the Repository with Rules SDK .......... 2-26
2.9.3 Loading a Dictionary with Rules SDK .......... 2-27
2.9.4 Specifying a Rule Set and Generating RL with Rules SDK .......... 2-27
2.9.5 Initializing and Executing a Rule Session .......... 2-28
2.9.6 Asserting Business Objects Within a Rule Session .......... 2-28
2.9.7 Using the Run Function with a Rule Session .......... 2-29
2.10 Running the Car Rental Sample Using the Test Program .......... 2-29

## 3 Working With Rule Author Features

3.1 Working with Variables .......... 3-1
3.2 Working with Constraints .......... 3-2
3.3 Working with RLFacts .......... 3-5
3.4 Working with Functions .......... 3-6
3.5 Viewing Java Objects in a Data Model .......... 3-8
3.5.1 Specifying Visibility and Object Chaining for Rule Author Drop Down Lists .......... 3-9
3.6 Generating Oracle Business Rules RL Language Text .......... 3-10
3.6.1 Generating and Viewing an RL Language Program .......... 3-10
3.7 Configuring Rule Author Dictionary Properties .......... 3-10
3.7.1 Using the Advanced Test Expression Option .......... 3-11
3.7.2 Using the Logging Option .......... 3-12
3.8 Deleting a Rule Author Dictionary .......... 3-12
3.9 Importing and Exporting a Dictionary .......... 3-13
3.10 Working with Test Rulesets .......... 3-14
3.11 Invoking Rules .......... 3-17
3.11.1 Overview of Results Examples .......... 3-18
3.11.2 Using a Global Variable to Obtain Results .......... 3-18
3.11.3 Using Container Objects to Obtain Results .......... 3-19
3.11.4 Using Reasoned On Objects to Obtain Results .......... 3-20

## 4 Using XML Facts with Rule Author

4.1 Overview of Using XML Documents and Schemas with Rule Author .......... 4-1
4.2 Creating and Saving a Dictionary for the XML Car Rental Sample .......... 4-2
4.2.1 Connecting to a Rule Author Repository .......... 4-2
4.2.2 Creating a Rule Author Dictionary .......... 4-3

4.2.3 Saving a Rule Author Dictionary with a Version ..... 4-4
4.2.4 Saving a Rule Author Dictionary ..... 4-5
4.3 Defining a Data Model for the XML Car Rental Sample ..... 4-6
4.3.1 Using XML Schema as Facts in the XML Car Rental Sample ..... 4-6
4.3.2 Adding XML Facts for the Car Rental Sample (XML Schema Processing) ..... 4-6
4.3.3 Importing XML Schema Elements to a Data Model ..... 4-9
4.3.4 Viewing XML Facts in a Data Model ..... 4-11
4.3.5 Saving the Current State of XML Fact Definitions ..... 4-11
4.4 Defining Business Vocabulary for the XML Car Rental Sample ..... 4-11
4.4.1 Specifying the Business Vocabulary for XML Fact Definitions ..... 4-12
4.4.2 Specifying the Business Vocabulary for Functions ..... 4-12
4.5 Defining a Rule for the XML Car Rental Sample ..... 4-13
4.5.1 Creating a Rule Set for the XML Car Rental Sample ..... 4-13
4.5.2 Creating a Rule for the XML Car Rental Sample ..... 4-14
4.6 Customizing Rules for the XML Car Rental Sample ..... 4-22
4.7 Creating a Java Application with a Rule Session Using XML Facts ..... 4-23
4.7.1 Importing the Rules SDK and Rules RL Classes ..... 4-24
4.7.2 Creating a JAXB Context and Unmarshalling the XML Document ..... 4-24
4.7.3 Loading a Dictionary with Rules SDK ..... 4-25
4.7.4 Loading a Ruleset and Generating RL Language for Data Model and Rule Set ..... 4-25
4.7.5 Initializing and Executing a Rule Session ..... 4-26
4.7.6 Asserting XML Data from Within a Rule Session ..... 4-26
4.7.7 Using the Run Function with a Rule Session ..... 4-27
4.8 Running the XML Car Rental Sample Using the Test Program ..... 4-27

## 5 Using JSR-94

5.1 Oracle Business Rules with JSR-94 Rule Execution Sets ..... 5-1
5.1.1 Creating a JSR-94 Rule Execution Set from Rule Sets in a File Repository ..... 5-1
5.1.2 Creating a JSR-94 Rule Execution Set from a WebDAV Repository ..... 5-2
5.1.3 Creating a Rule Execution Set from Oracle Business Rules RL Language Text ..... 5-3
5.1.4 Creating a Rule Execution Set from RL Text Specified in a URL ..... 5-5
5.1.5 Creating Rule Execution Sets with Rule Sets from Multiple Sources ..... 5-6
5.2 Using the JSR-94 Interface with Oracle Business Rules ..... 5-6
5.2.1 Creating a Rule Execution Set with CreateRuleExecutionSet ..... 5-6
5.2.2 Creating a Rule Session with createRuleSession ..... 5-7
5.2.3 Working with JSR-94 Metadata ..... 5-7
5.2.4 Using Oracle Business Rules JSR-94 Extensions ..... 5-8

## 6 Using Oracle Business Rules SDK

6.1 Rules SDK building blocks ..... 6-1
6.2 Working with a Repository and a Dictionary ..... 6-2
6.2.1 Establishing Contact with a WebDAV Repository ..... 6-2
6.2.2 Establishing Contact with a File Repository ..... 6-3
6.2.3 Loading a Dictionary ..... 6-3
6.3 Working with a Data Model ..... 6-3
6.3.1 Creating a DataModel ..... 6-4

6.3.2 Creating DataModel Components .......... 6-4
6.3.3 Creating a Function Argument List .......... 6-5
6.3.4 Creating an Initializing Expression .......... 6-5
6.3.5 Creating RL Function Bodies .......... 6-6
6.4 Using RuleSets and Creating and Modifying Rules .......... 6-6
6.4.1 Creating a RuleSet .......... 6-7
6.4.2 Adding a Rule to a Ruleset .......... 6-7
6.4.3 Adding a Pattern to a Rule .......... 6-8
6.4.4 Adding a Test to a Pattern .......... 6-8
6.4.5 Adding an Action to a Rule .......... 6-9
6.4.6 Notes for Adding RuleSets and Rules .......... 6-10

## A Oracle Business Rules Files and Limitations

A.1 Rule Author Naming Conventions .......... A-1
A.1.1 Ruleset Naming .......... A-1
A.1.2 Dictionary Naming .......... A-1
A.1.3 Version Naming .......... A-1
A.1.4 Alias Naming .......... A-1
A.1.5 XML Schema Target Package Naming .......... A-2
A.2 Rule Author Session Timeout .......... A-2
A.3 Rules SDK and Rule Author Temporary Files .......... A-2

## B Using Rule Author and Rules SDK with Repositories

B.1 Working with a WebDAV Repository .......... B-1
B.1.1 Setting up a WebDAV Repository .......... B-1
B.1.2 Connecting to a WebDAV Repository .......... B-2
B.2 WebDAV Repository Security .......... B-2
B.2.1 Communicating with a WebDAV Repository Over SSL from Rule Author .......... B-3
B.2.2 Setting the Location of your Oracle Wallet .......... B-3
B.2.3 Configuring Rule Author for WebDAV Repository Authentication .......... B-3
B.2.4 Storing Data in an Oracle Wallet for WebDAV Repository Authentication .......... B-4
B.3 Working with a File Repository .......... B-5
B.3.1 Setting up a File Repository .......... B-5
B.3.2 File Repository Updates and Temporary Files .......... B-5
B.4 High Availability for your Repository .......... B-6

## C Oracle Business Rules Frequently Asked Questions

C.1 Frequently Asked Questions About Rules Operations .......... C-1
C.1.1 Why is the State of a Fact in a Rule Action Inconsistent with the Rule Condition? . C-1
C.1.2 A Changed Java Object was Asserted as a Fact, but no Rules Fired. Why? .......... C-2
C.1.3 What are the Differences Between Oracle Business Rules RL Language and Java?. C-2
C.2 What JAR Files are Required for Working with Oracle Business Rules? .......... C-2

## D Oracle Business Rules Troubleshooting

D.1 Public Fact Variables are not Accessible with Rule Author .......... D-1
D.2 Global Variables may not be Used in RL Functions .......... D-2

D.3 Importing JDK 1.4.2 Classes ..... D-2
D.4 Managing Popup Windows on Firefox ..... D-2
D.5 Using the String Data Type with Methods..... D-2
D.6 Preserving Class Order and Hierarchies in the Data Model ..... D-2
D.7 Validating Generated RL from Rule Author..... D-3
D.8 Using RL Reserved Words as Part of a Java Package Name ..... D-3
D.9 Getter and Setter Methods are not Visible ..... D-3
D.10 XML Facts not Asserted at Runtime ..... D-3

Index

# Preface

This Preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

## Audience

Oracle Business Rules User's Guide is intended for application programmers, system administrators, and other users who perform the following tasks:

- Create Oracle Business Rules programs
- Modify or customize existing Oracle Business Rules programs
- Create new Java applications using rules programs
- Add rules programs to existing Java applications

To use this document, you need a working knowledge of Java programming language fundamentals.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

## Related Documentation

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
|---|---|
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

# 1 Overview of Oracle Business Rules

This guide provides information on using Oracle Business Rules. Oracle Business Rules is a component of Oracle Application Server that enables applications to rapidly adapt to regulatory and competitive pressures. This increased agility is possible because business analysts using Oracle Business Rules can create and change business rules that are separated from the application code. Using Oracle Business Rules, business rules can change without stopping business processes. Also, externalizing business rules allows business analysts to manage business rules directly, without involving programmers.

This guide shows you how to work with Oracle Business Rules Rule Author (Rule Author), describes the Oracle Business Rules SDK (Rules SDK) and describes how to create a rule enabled Java program.

This chapter covers the following topics:

- Introduction to Oracle Business Rules
- Oracle Business Rules Components
- Oracle Business Rules Rule Author Terms and Concepts
- Steps for Rule Enabling a Java Application

# 1.1 Introduction to Oracle Business Rules

This section introduces the concept of business rules and covers the following:

- What Are Business Rules?
- What Is a Data Model?
- What Is a Rule-Based System?

## 1.1.1 What Are Business Rules?

Business rules are statements that describe business policies. For example, a car rental company might use the following business rule:

If a driver's age is younger than 21, then decline to rent.

An airline might use a business rule such as the following:

If a frequent flyer's total miles for the year are greater than 100,000, then status is Gold.

A financial institution could use a business rule such as:

If annual income is less than $10,000, then deny loan.

These examples represent individual business rules. In practice, using Oracle Business Rules you can combine many business rules.

For the car rental example, you can name the driver age rule the Under Age rule. Traditionally, business rules such as the Under Age rule are buried in application code, and might appear in a Java application as follows:

```
public boolean checkUnderAgeRule (Driver driver) {
   boolean declineRent = false;
   int age = driver.getAge();
   if(  age < 21 ) {
      declineRent = true;
   }
   return declineRent;
}
```

This code is not easy for non-technical users to read and can be difficult to understand and modify. For example, suppose that the rental company changes its policy to "Under 18", so that all drivers under 18 match for the Under Age rule. In many production environments, the developer would need to modify the application, recompile, and then redeploy the application. Using Oracle Business Rules, this process can be simplified.

Oracle Business Rules allows a business analyst to change business policies that are expressed as rules, with little or no assistance from a programmer. Applications using Oracle Business Rules, called rule enabled applications, can quickly adapt to new government regulations, improvements in internal company processes, or changes in relationships between customers and suppliers.

**See Also:** "Steps for Rule Enabling a Java Application" on page 1-9

## 1.1.2 What Is a Data Model?

In Oracle Business Rules, **facts** are data objects that are asserted in the Rules Engine. Rules, such as the Under Age rule operate on facts. In Oracle Business Rules, a **data model** specifies the types of facts or business objects that you can use when you create business rules. For example, for a car rental company that needs to create a rule to match a driver's age, the driver's age information represents the facts used in the rule.

Using Rule Author, you can create a data model and then use the objects in the data model when you create rules.

## 1.1.3 What Is a Rule-Based System?

This section covers the following:

- Rule-Based Systems Using the Rete Algorithm
- Oracle Business Rules Rule-Based Systems

### 1.1.3.1 Rule-Based Systems Using the Rete Algorithm

The Rete algorithm was first developed by artificial intelligence researchers in the late 1970s and is at the core of Rules Engines from several vendors. Oracle Business Rules uses the Rete algorithm to optimize the pattern matching process for rules and facts. The Rete algorithm stores partially matched results into a single network of nodes in current working memory.

Using the Rete algorithm, the Rules Engine avoids unnecessary rechecking when facts are deleted, added, or modified. To process facts and rules using the Rete algorithm, there is an input node for each fact definition and there is an output node for each rule. Fact references flow from input to output nodes[1].

The Rete algorithm provides the following benefits:

- Independence from rule order – rules can be added and removed without impacting other rules.
- Optimization across multiple rules – rules with common conditions share nodes in the Rete network.
- High performance inference cycles – each rule firing typically changes just a few facts and the cost of updating the Rete network is proportional to the number of changed facts, not to the total number of facts or rules.

### 1.1.3.2 Oracle Business Rules Rule-Based Systems

A rule-based system using the Rete Algorithm is the foundation of Oracle Business Rules. A rule-based system consists of the following:

- The Rule-base: contains the appropriate business policies or other knowledge encoded into If-Then rules.
- Working memory: contains the information that has been added to the system. Using Oracle Business Rules; this consists of a set of facts.
- Inference Engine: the Rules Engine which processes the rules performs pattern matching to determine which rules match the facts, for a given run through the set of facts.

Using Oracle Business Rules the rule-based system is a data-driven **forward chaining system**. The facts determine which rules can fire, when a rule fires that matches a set of facts, the rule may add new facts which are once again run against the rules. This process repeats until a conclusion is reached or the cycle is stopped or reset. Thus, in a

[1] Fact references flow from input to output nodes. In between input and output nodes are test nodes and join nodes. A test occurs when a rule condition has a boolean expression. A join occurs when a rule condition ANDs two facts. A rule is activated when its output node contains fact references. Fact references are cached throughout the network to speed up recomputing activated rules. When a fact is added, removed, or changed, the Rete network updates the caches and the rule activations with only an incremental amount of work.

forward chaining rule-based system, facts cause rules to fire, and firing rules can create more facts, which in turn can fire more rules. This process is called an inference cycle.

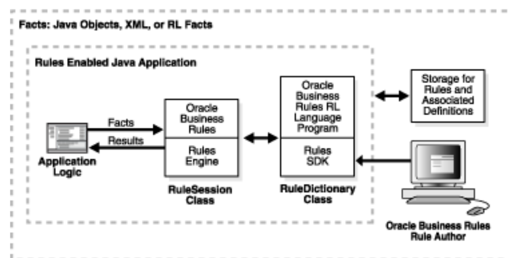
## 1.2 Oracle Business Rules Components

Figure 1–1 shows the Oracle Business Rules components.

This section covers the following topics:

- Introducing Oracle Business Rules Rule Author
- Introducing Oracle Business Rules Rules SDK
- Introducing Oracle Business Rules RL Language
- Introducing Oracle Business Rules Rules Engine

***Figure 1–1 Oracle Business Rules Architecture***




### 1.2.1 Introducing Oracle Business Rules Rule Author

Oracle Business Rules Rule Author (Rule Author) lets you work with rules from anywhere using a web browser and provides a point-and-click interface for creating new rules and editing existing rules. Rule Author reduces a rule developer's work, allowing you to work directly with business rules and a data model. You do not need to understand the Oracle Business Rules RL Language (RL Language) to work with Rule Author. Rule Author provides an easy way for business users to create, view, and modify business rules.

Rule Author supports several types of users, including the application developer and the business analyst. The application developer uses Rule Author to define a data model and an initial set of rules. The business analyst uses Rule Author to either work with the initial set of rules, or modify and customize the initial set of rules according to business needs. Using Rule Author, a business analyst can create and customize rules with little or no assistance from a programmer.

Rule Author stores rules programs in a dictionary that is saved to a repository using a Rule Author dictionary storage plug-in. You can create as many dictionaries as necessary, and each dictionary can have multiple versions. A rule enabled program accesses a dictionary with the Oracle Business Rules SDK.

**Note:** It is not safe for multiple users to edit the same dictionary.

As shipped, Rule Author supports a WebDAV (Web Distributed Authoring and Versioning) repository and a file repository.

---

**Note:** For file repositories, only one user may edit the repository at any given time, regardless of the number of dictionaries stored in the repository. For WebDAV repositories, a single user may edit multiple dictionaries simultaneously.

---

### 1.2.2 Introducing Oracle Business Rules Rules SDK

Oracle Business Rules SDK (Rules SDK) is a Java library that provides business rule management features that a developer can use to write customized rules programs. Rule Author uses the Rules SDK to create, modify, and access rules and the data model using well defined interfaces. Customer applications can use the Rules SDK to display, create, and modify collections of rules and the data model.

You can use the Rules SDK APIs in a rule enabled application to access, or to create and modify rules (the rules and the associated data model could be initially created in a custom application or using Rule Author where they are stored using a Rules SDK dictionary storage plug-in).

Using the Rules SDK you can also support custom repositories, using the dictionary storage plug-in portion of the Rules SDK API.

### 1.2.3 Introducing Oracle Business Rules RL Language

Oracle Business Rules supports a high level Java-like language called Oracle Business Rules RL Language (RL Language). The RL Language defines the valid syntax for Oracle Business Rules programs. RL Language includes an intuitive Java-like syntax for defining rules that supports the power of Java semantics, providing an easy-to-use syntax for application developers. RL Language consists of a collection of text statements that can be generated dynamically, or stored in a file.

Using RL Language, application programs can assert Java objects as facts, and rules can reference object properties and invoke methods. Likewise, application programs can use XML documents, or portions of XML documents as facts.

Programmers can use RL Language as a full-featured rules programming language. Business analysts can use Rule Author to work with rules, and in this case the business analyst does not need to directly view or write RL Language programs.

**See Also:** *Oracle Business Rules Language Reference Guide* for detailed information on RL Language.

### 1.2.4 Introducing Oracle Business Rules Rules Engine

The Oracle Business Rules Rules Engine (Rules Engine) is a Java library that efficiently applies rules to facts and defines and processes rules. The Rules Engine defines a declarative rule language, provides a language processing engine (inference engine), and provides tools to support debugging.

The Rules Engine has the following features:

- High performance – implementing specialized matching algorithms.
- Thread-safe execution suitable for a parallel processing architecture (one thread can assert facts while another is evaluating the network).
- Agility – meaning business rules can change without stopping business processes.

A rule enabled Java application can load and run rules programs. The rule enabled application passes facts, in the form of Java objects or XML documents, and rules to

the Rules Engine. The Rules Engine runs in the rule enabled Java application, and uses the Rete algorithm to efficiently fire rules that match the facts.

The Rules Engine supports an interactive command-line interface for development, testing, and debugging of RL Language programs.

## 1.3 Oracle Business Rules Rule Author Terms and Concepts

This section provides information on Rule Author terms and concepts and covers the following topics:


- Working with Rules
- Working with Rule Sets
- Working with Repositories and Dictionaries
- Working with Facts
- Working with Functions Variables and Constraints


### 1.3.1 Working with Rules

A rule consists of a condition, or **If** part, and a list of actions, or a **Then** part. Rules follow a simple if-then structure. This section covers the components of a rule.

#### 1.3.1.1 Rule Conditions

The rule **If** part is composed of conditional expressions, rule conditions, that refer to facts. For example,

If a driver's age is younger than 21.

The conditional expression refers to a fact (driver), followed by a test that the fact's data member, age, is less than 21.

The rule condition activates the rule whenever there is a combination of facts that makes the conditional expression true. In some respects the rule condition is like a query over the available facts in the Rules Engine, and for every row returned from the query, the rule is activated.

#### 1.3.1.2 Rule Actions

The rule **Then** part contains the actions that are executed if all of the rule conditions are satisfied. The actions are executed, or fired, when all of the conditions in the **If** part are met. There are several kinds of actions that a rule might perform. An action can add new facts or remove facts. An action can execute a Java method or perform an RL Language function, which may modify the status of facts or create new facts.

Rules fire sequentially, not in parallel. Note that rule actions often change the set of rule activations and thus change the next rule to fire.

**See Also:** "Working with Rule Sets" on page 1-6

### 1.3.2 Working with Rule Sets

A rule set groups a set of rules. A rule set is a collection of rules that are all intended to be evaluated together.

**See Also:** "Working with Rules" on page 1-6

### 1.3.3 Working with Repositories and Dictionaries

Using Oracle Business Rules, a repository stores dictionaries. A dictionary usually corresponds to a rules application. A dictionary is a set of XML files that stores the application's rule sets and the data model. You store a dictionary in a repository using a supplied dictionary storage plug-in or a custom dictionary storage plug-in. The dictionary storage plug-in API is part of the Rules SDK. A dictionary typically stores all the rules and definitions for a rule enabled application. Dictionaries my have different versions. Dictionaries and dictionary versions can be created, deleted, exported, and imported into a repository.

As shipped, Rule Author supports a WebDAV (Web Distributed Authoring and Versioning) repository and a file repository.

### 1.3.4 Working with Facts

Using Rule Author, facts are data objects that have been asserted in the Rules Engine. Each object instance corresponds to a single fact. If an object is re-asserted (whether it has been changed or not), the Rules Engine is updated to reflect the new state of the object. Re-asserting the object does not create a new fact. In order to have multiple facts of a particular fact type, separate object instances must be asserted.

Using Rule Author, you can create rules that operate on facts that are part of a data model. You make business objects and their methods known to Oracle Business Rules using fact definitions.

This section covers the three types of Oracle Business Rules fact definitions:

- Java Fact Type Definitions
- XML Fact Type Definitions
- Oracle Business Rules RL Language Fact Type Definitions

You typically use Java Fact Types and XML Fact Types to create rules that examine the business objects in a rule enabled application, or to return results to the application and you use RL Language Fact Type definitions to create intermediate facts that can trigger other rules in the Rules Engine.

#### 1.3.4.1 Java Fact Type Definitions

A Java fact type allows selected properties and methods of a Java class to be declared to the Rules Engine so that rules can access, create, modify, and delete instances of the Java class. Declaring a Java fact type allows the Rules Engine to access and use public attributes, public methods, and bean properties defined in a Java class (bean properties are preferable for some applications because the Rules Engine can detect that a Java object supports `PropertyChangeListener`; in this case it utilizes that mechanism to be notified when the object changes).

#### 1.3.4.2 XML Fact Type Definitions

An XML fact type allows selected attributes and sub-elements of an XML element or `complexType` to be declared to the Rules Engine so that instances of it can be accessed, created, modified, and deleted by rules.

> **See Also:** "Overview of Using XML Documents and Schemas with Rule Author" on page 4-1

#### 1.3.4.3 Oracle Business Rules RL Language Fact Type Definitions

An RL Language fact type is similar to a relational database row or a Java Bean without methods. An RL Language fact type contains a list of members of either RL Language fact type, Java fact type, or primitive type. RL Language fact types can be used to extend a Java application's object model by providing virtual dynamic types.

For example,

If customer spent $500 within past 3 months

then customer is a Gold Customer

This rule might use a Java fact type, specifying the customer data and also use an action that creates an RL Language fact type, Gold Customer. A rule might be defined to use a Gold Customer fact, as follows:

If customer is a Gold customer

then offer 10% discount

This rule uses the RL Language fact type, named Gold Customer. This rule then infers, using the Gold Customer fact, that if a customer spent $500 within the past 3 months, then the customer is eligible for a 10% discount. In addition, there could be other ways specified in the rules that a customer becomes a Gold Customer.

### 1.3.5 Working with Functions Variables and Constraints

This section covers the following definitions:

- Function Definitions
- Constraint Definitions
- Variable Definitions

#### 1.3.5.1 Function Definitions

Using Oracle Business Rules, a function is similar to a Java method, but it does not belong to a class. You can use functions to extend a Java application object model so that users can perform operations in rules without modifying the original Java application code.

You can also use a function definition to share the same or a similar expression among several rules, and to return results to the application.

#### 1.3.5.2 Constraint Definitions

Constraint definitions let you mark portions of rules as customizable. For example, the discount to offer to a Gold customer could be constrained to be within a specified range such as 5 to 25 percent. Using Rule Author, by defining a constraint, you can select a value from within the specified range using a special interface that does not allow you to modify the entire rule.

> **Note:** Use of constraints is a Rule Author feature that supports rule customization (using the Rule Author rule customization tab).

#### 1.3.5.3 Variable Definitions

You can use variable definitions to share information among several rules and functions. For example, if a 10% discount is used in several rules, you can create and

use a variable Gold Discount, so that the appropriate discount is applied to all the rules using the variable.

Using variable definitions can make programs modular and easier to maintain.

## 1.4 Steps for Rule Enabling a Java Application

Programmers and business analysts work together to rule-enable a Java application. For many applications, after the application is rule enabled, the programmer role diminishes over time, leaving ongoing rule maintenance to the business analyst.

The tasks required to rule-enable a Java application include:

- Identify Application Areas to Rule Enable
- Provide Rule Author Definitions for the Data Model
- Develop a Business Vocabulary for the Data Model
- Write and Customize Rules
- Modify or Create Application Logic that Uses the Rules Engine
- Test the Rule Enabled Application

These tasks require cooperation between the programmer and the business analyst. Programmers understand application code and are comfortable with Java development, web services, and XML (if the business objects are represented in XML). Business analysts understand the business objects at a higher level, and the business analysts should understand rules as if…then statements concerning business objects. The business analysts also need to determine the parts of rules that are likely to need frequent change.

### 1.4.1 Identify Application Areas to Rule Enable

The business analyst and programmer collaborate to expose business objects as facts suitable for use in business rules. The business analyst determines the business facts required for use with the business rules.

The business analyst should determine what functionality should be rule-driven. For example, in an online shopping application, perhaps the tax and promotion functions should be rule-based, but not the shopping cart or product catalog.

### 1.4.2 Provide Rule Author Definitions for the Data Model

The programmer uses definitions in Rule Author to specify the data model (working with the Rule Author Definitions tab). Working with the business analyst the programmer also defines helpful functions, intermediate facts, variables, and constraints.

### 1.4.3 Develop a Business Vocabulary for the Data Model

The programmer and the business analyst use Rule Author to develop a business-friendly vocabulary for the Rule Author definitions, so that the rules are more understandable. While determining what business facts, functions, and other definitions to capture, the business analyst has been developing a vocabulary. The programmer enters this information using Rule Author.

### 1.4.4 Write and Customize Rules

At this point, the business analyst should be able to use Rule Author to write and customize rules using the defined business vocabulary. Alternatively, the programmer may use the Rules SDK to create or modify rules, or the data model from within the administrative portion of a rules-enabled application.

### 1.4.5 Modify or Create Application Logic that Uses the Rules Engine

The programmer determines how to replace procedural functionality with new rule-driven functionality. If the application is written in Java then the application code can directly invoke the Rules Engine. Otherwise, the programmer may need to invoke the Rules Engine using a web service or other remote API. The programmer must either create a new application or modify an existing application to interact with the Rules Engine.

---

**Note:** Procedural code that is being rule enabled may need to be "mined" to extract existing hard-coded rules.

---

**See Also:** Chapter 2, "Getting Started with Rule Author" for details on working with Rule Author and on the steps a programmer takes to rule-enable a Java application.

### 1.4.6 Test the Rule Enabled Application

The programmer and the business analyst test the modifications made to the application. The programmer may need to assist in the debugging of a complex set of rules. Rules Engine tracing can be enabled to provide information about facts, rule activations, and rule firings. A mechanism for loading test facts should be developed that validates a set of the business analyst rules.

# 2 Getting Started with Rule Author

This chapter provides a tutorial introducing Oracle Business Rules Rule Author (Rule Author). This chapter shows you how to start Rule Author, create a data model, and create and save rules. This chapter also shows you how to create a sample Java application that runs with the Rules Engine.

In this guide we use a car rental sample to illustrate how to work with Rule Author. In the car rental sample, driver data specifies driver information and the business rules determine if a rental company service representative should decline to rent a vehicle due to driver age restrictions. Using this example you create one rule, the UnderAge rule (the rule is specified according to rental company business rules).

This chapter covers the following:


- Creating a Rule Author User
- Starting Rule Author
- Rule Author Home Page
- Creating and Saving a Dictionary for the Car Rental Sample
- Defining a Data Model for the Car Rental Sample
- Defining Business Vocabulary for the Car Rental Sample
- Defining a Rule for the Car Rental Sample
- Customizing Rules for the Car Rental Sample
- Creating a Java Application Using Oracle Business Rules
- Running the Car Rental Sample Using the Test Program


## 2.1 Creating a Rule Author User

If you are using Oracle Application Server, you must first create a user with appropriate privileges before you can start and use Rule Author. To do this:

---

**Note:** These instructions assume that the container is configured with the JAZN XML provider. If it is not, you should refer to the appropriate security documentation for information on creating users.

---

1. Using Application Server Control, go to the OC4J instance where Rule Author was deployed.
2. Click the **Administration** tab.

3. In the "Task Name" column, find the "Security Providers" task and click the "Go to Task" icon in the corresponding row.
4. Click **Instance Level Security**.
5. Click the **Realms** tab.
6. In the table in the "Results" section, click the number in the "Roles" column to add a role.
7. Click the **Create** button.

   In the "Name" field, enter `rule-administrators`, then click **OK**.
8. Click the **Instance Level Security** link near the top of the page (in the navigation trail) to return to the Instance Level Security page.
9. In the table in the "Results" section, click the number in the "Users" column to add a user.
10. Click the **Create** button.

    In the "Name" field, enter the name you want to use to login to Rule Author (for example, `ruleadmin`). Enter and confirm the password for this user. In the "Assign Roles" section, double-click or use the arrows to assign the `rule-administrators` role to this user. When you are finished, click **OK**.

    > **Note:** In order to Rule Author authentication to work, the Rule Author user must belong to the `rule-administrators` role.

11. Restart the Rule Author application.

## 2.2 Starting Rule Author

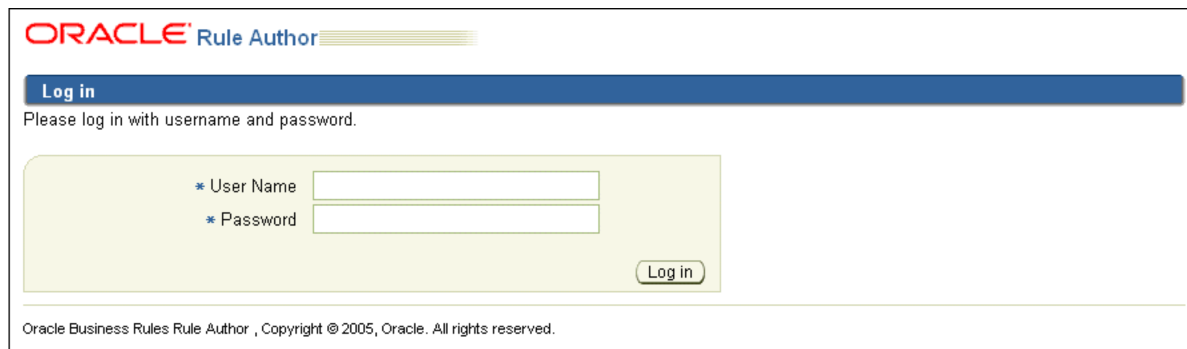
Start Rule Author by entering the URL for the home page. The URL for the home page typically includes the name of the host computer and the port number assigned to the application server during the installation, plus the path of the Rule Author home page. For example:

```
http://myhost1.mycompany.com:7777/ruleauthor/
```

> **Note:** The port number assigned to Oracle Application Server can be found in the `readme.txt` file located in the *`$ORACLE_HOME`*`/install` directory.

Figure 2–1 shows the Rule Author login page. Specify the user name and password you supplied when you created the Rule Author user (Section 2.1, "Creating a Rule Author User").

***Figure 2–1 Rule Author Login Page***




After logging in, you should see the Rule Author Repository Connect page (see Figure 2–2). You must connect to a repository before you can perform any operations. See Section 2.4.1, "Connecting to a Rule Author Repository" for more information.

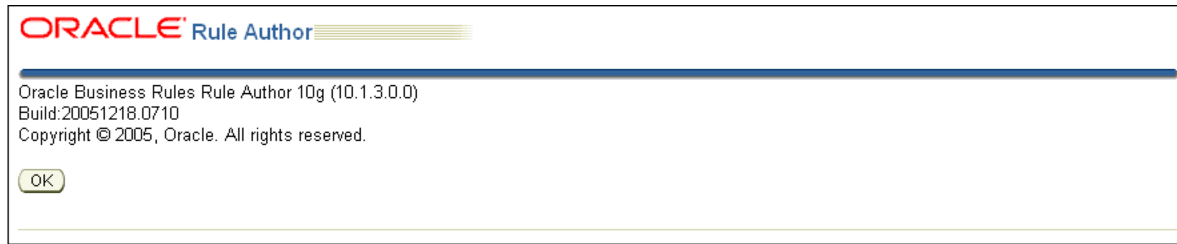
***Figure 2–2 Initial Rule Author Repository Connect Page***

Click **Logout** to go to the Logout Confirmation page. On this page, click either **Logout** to log out of Rule Author, or **Save and Logout** to save your changes and log out of Rule Author. After doing so, you must log back in to Rule Author (Figure 2–1).

Click **Help** to access online help for the Rule Author.

Click **About** to view version and build information about the Rule Author (Figure 2–3). Click **OK** to dismiss this window.

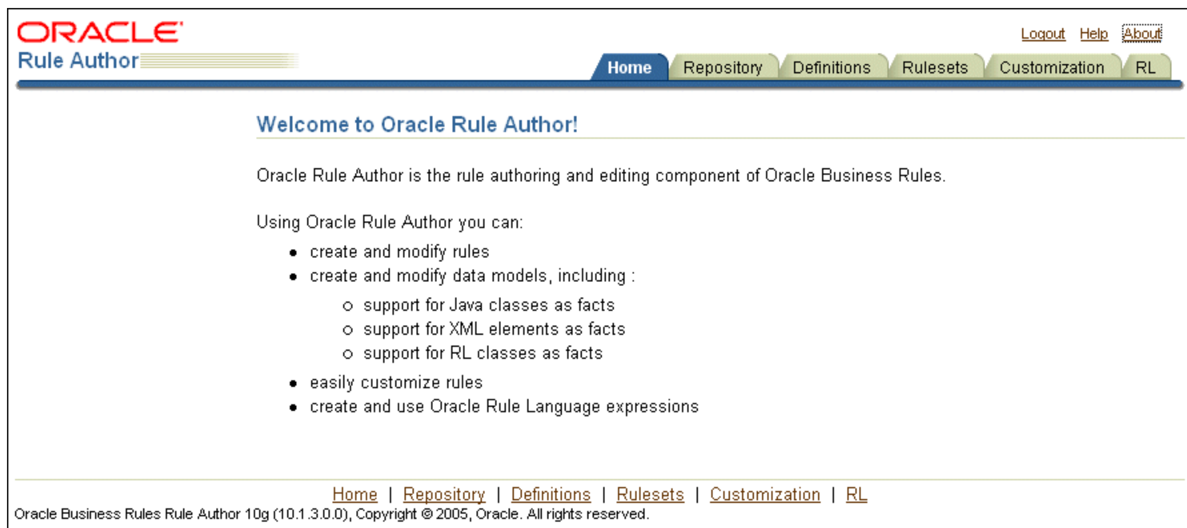
***Figure 2–3 About Rule Author***




## 2.3 Rule Author Home Page

Click the **Home** tab to access the Rule Author home page (Figure 2–4). The home page contains two panes: the top pane shows the tabs and the bottom pane contains content for the currently selected tab.

***Figure 2–4 Rule Author Home Page***




## 2.4 Creating and Saving a Dictionary for the Car Rental Sample

To work with Rule Author you need to start with a dictionary. Rule Author stores rules and their associated definitions in a dictionary. To create or save a dictionary, you must first connect to a repository that stores the dictionary. As shipped, Rule Author supports two types of repositories: WebDAV (Web Distributed Authoring and Versioning) repository and file repository. In this section you create and save a dictionary for the car rental How-To.

The example in this chapter saves the dictionary to a WebDAV repository.

**Note:** To create the dictionary shown in this chapter, you can either create a new dictionary, using either a WebDAV repository or a file repository, or you can load the completed dictionary from the `CarRepository` file repository in the `/dict` directory supplied with the car rental sample How-To.

See Section 2.4.2, "Creating a Rule Author Dictionary" for instructions on how to do this.

### 2.4.1 Connecting to a Rule Author Repository

Using Oracle Business Rules, a dictionary stores rules and the data model associated with the rules. You create and save dictionaries in a repository.

**Note:** Regardless of whether you choose to use a WebDAV or file repository, the repository must exist before you can connect to it. Rule Author does not create the repository for you.

See Appendix B, "Using Rule Author and Rules SDK with Repositories" for more information.

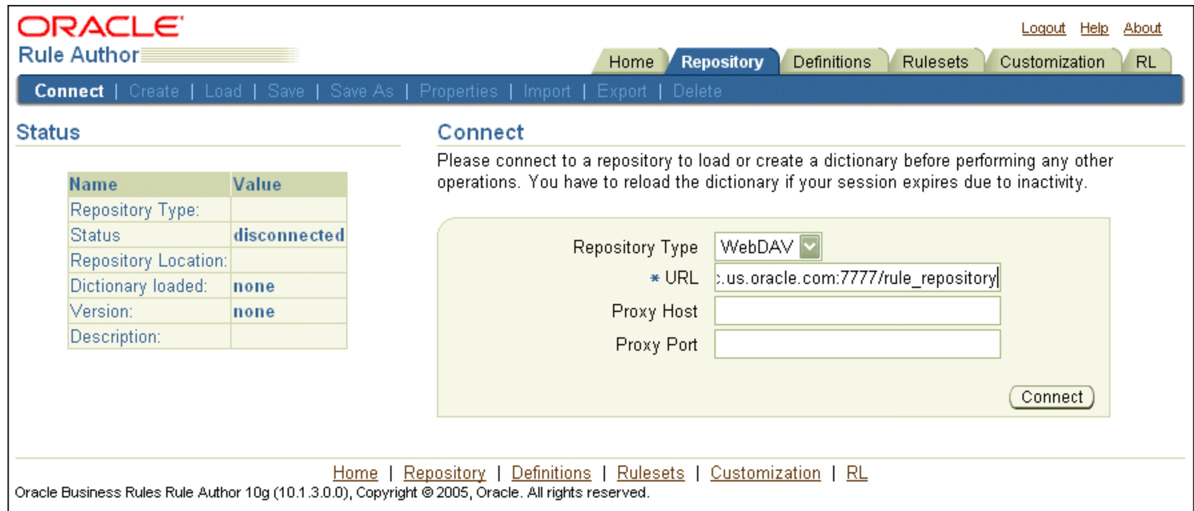
To connect to a repository, do the following:

1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. Select the **WebDAV** repository type in the **Repository Type** field.
4. Enter the URL to the WebDAV repository (see Figure 2–5). The URL must be in the form:

   ```
   http://www.fully_qualified_host_name.com:7777/repository_name
   ```

**Note:** In order for authentication to work, you must use a fully qualified host name in the URL.

***Figure 2–5 Rule Author WebDAV Repository Connect Page***




See Section B.1, "Working with a WebDAV Repository" for information on how to setup a WebDAV repository.

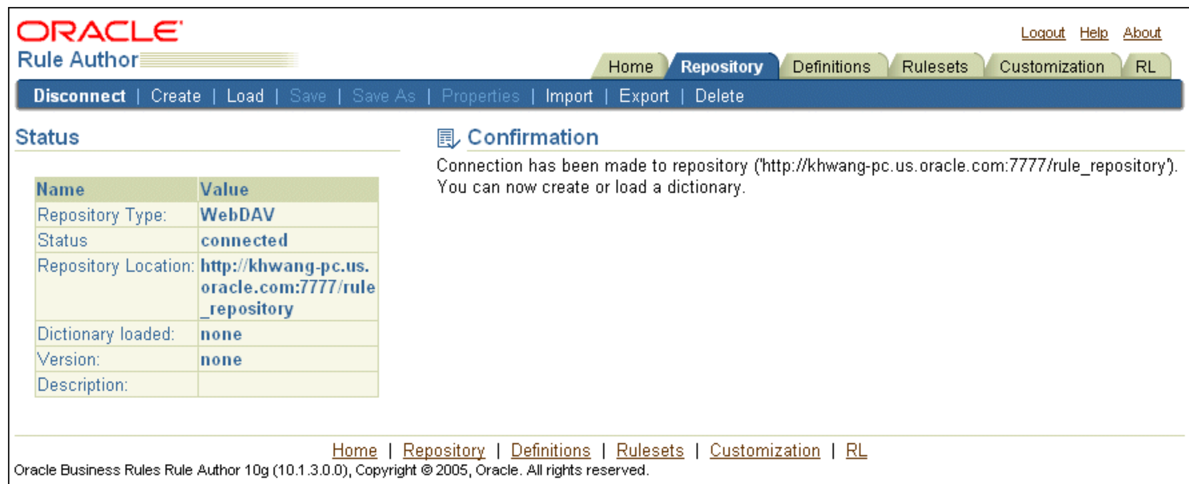
5. If you have a proxy server between the server on which Rule Author is running and the WebDAV server, specify the name and port number of the proxy server.

6. Click **Connect**.

   If you connect successfully, a confirmation message is displayed (see Figure 2–6).

> **Note:** For file repositories, only one user may edit the repository at any given time, regardless of the number of dictionaries stored in the repository. For WebDAV repositories, a single user may edit multiple dictionaries simultaneously.

***Figure 2–6 Rule Author Repository Connect Page with Confirmation***



## 2.4.2 Creating a Rule Author Dictionary

A Rule Author dictionary is the top-level container and the starting point for working with Rule Author. A dictionary usually corresponds to the rules portion of an application.

---

**Note:** It is not safe for multiple users to edit the same dictionary.

---

To create a dictionary, do the following:

1. Connect to a repository from the **Repository** tab.
2. Click **Create**.
3. Enter the dictionary name in the **New Dictionary Name** field. For this example enter `CarRental` (see Figure 2–7).
4. Click **Create**. After you click **Create**, Rule Author shows a status message.

***Figure 2–7 Rule Author Create Dictionary Page***

---

**Note:** In addition to creating your own dictionary, you can also use the completed car rental dictionary that is supplied with the car rental sample How-To. This dictionary is located in the `CarRepository` file repository in the `/dict` directory:

1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. Select **File** in the **Repository Type** box.
4. Enter the complete path to the file repository in the **File Location** field. For example, enter `C:/demo/dict/CarRepository`.
5. Click **Connect**.

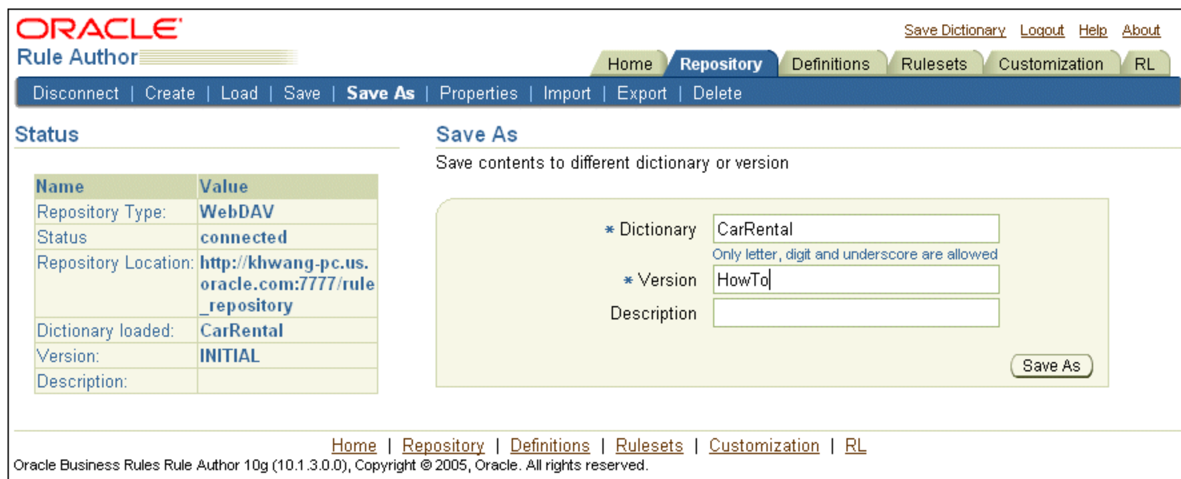
---

### 2.4.3 Saving a Rule Author Dictionary with a Version

If you want to save to a different dictionary name or specify a version for the current dictionary, use **Save As** as follows:

1. Click the **Repository** tab.
2. Click the **Save As** secondary tab.
3. Enter a dictionary name in the **Dictionary** field, for example `CarRental`.
4. To specify a version, enter a version in the **Version** field. For example `HowTo` (see Figure 2–8).
5. Click **Save As**. After clicking **Save As**, you should see a confirmation message.

***Figure 2–8 Rule Author Save As Page***




---

**Note:** Rule Author does not allow you to use **Save As** to overwrite a dictionary with the same name and version. If you want to overwrite a dictionary with the same name and version, do one of the following:

- Click **Save**.
- Delete the existing dictionary, then click the **Save As**.

---

### 2.4.4 Saving a Rule Author Dictionary

To prevent data loss, you should periodically save the dictionary. To save a dictionary, do one of the following:

- Click the **Repository** tab, then click the **Save** secondary tab.
- Click the **Save Dictionary** link at the top of the page.

After performing either of the preceding actions, click **Save** on the Save Dictionary page. After clicking **Save**, you should see a confirmation message in the status area. For example:

```
Dictionary 'CarRental(HowTo)' has been saved
```

---

**Note:** You should save the dictionary periodically as you work since Rule Author sessions time out after a period of inactivity.

---

**See Also:** "Rule Author Session Timeout" on page A-2 for details on configuring the Rule Author session timeout.

## 2.5 Defining a Data Model for the Car Rental Sample

Before working with rules you need to define a data model. A data model contains business data definitions for facts or data objects used in rules, including: Java class fact types, XML fact types, and RL Language fact types (to simplify the discussion in this section, we refer to Java fact types as Java facts). In this section you only work with Java facts.

This section covers the following topics:

- Using Java Objects as Facts in the Car Rental Sample
- Adding Java Classes and Packages to Rule Author
- Importing Java Classes to a Data Model
- Saving the Current State of Definitions

**See Also:**

"Importing XML Schema Elements to a Data Model" on page 4-9

### 2.5.1 Using Java Objects as Facts in the Car Rental Sample

The Java Car Rental How-To includes the `car-objs.jar` file in the *$HowToDir*`/lib` directory. This jar file includes the `Driver` class for the car rental sample. The Java source for the `Driver` object is available in the directory *$HowToDir*`/src/carrental`.

### 2.5.2 Adding Java Classes and Packages to Rule Author

Before you can import Java facts into a data model you need to make the classes and packages that contain the Java facts available to Rule Author. To do this use Rule Author to specify the classpath that contains the Java classes. For example, to add the classpath for the Java class `Driver`, do the following:

1. Click the **Repository** tab.
2. On the Load dictionary page, select the `CarRental` dictionary and `HowTo` version, then and click **Load** (skip this step if you just created the dictionary).
3. Click the **Definitions** tab. The navigation tree shows the Definitions folder that contains the available definitions. Nothing is shown in the main pane.

---

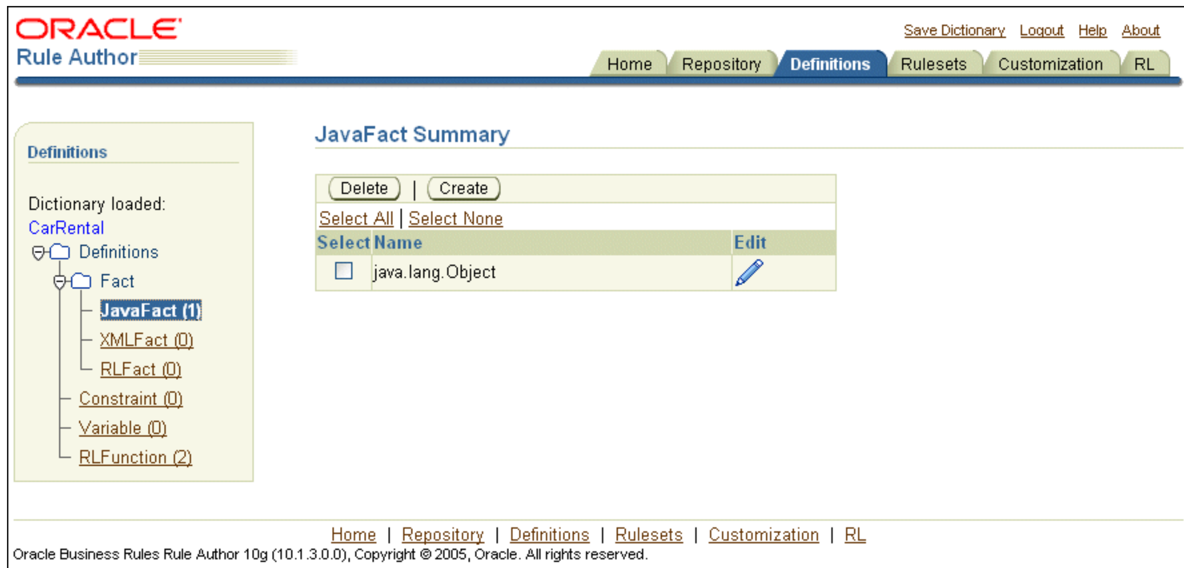
**Note:** Using Rule Author, the bottom pane usually contains a navigation tree and a content area (the main pane). With the **Definitions** tab selected, the Definitions folder is shown at the top of the tree.

---

4. The Definitions folder in the tree contains the Facts folder that includes the available fact types: **JavaFact**, **XMLFact**, and **RLFact**.

Click **JavaFact** to view the JavaFact Summary page (see Figure 2–9).

***Figure 2–9 Rule Author Definitions Java Fact Summary Page***

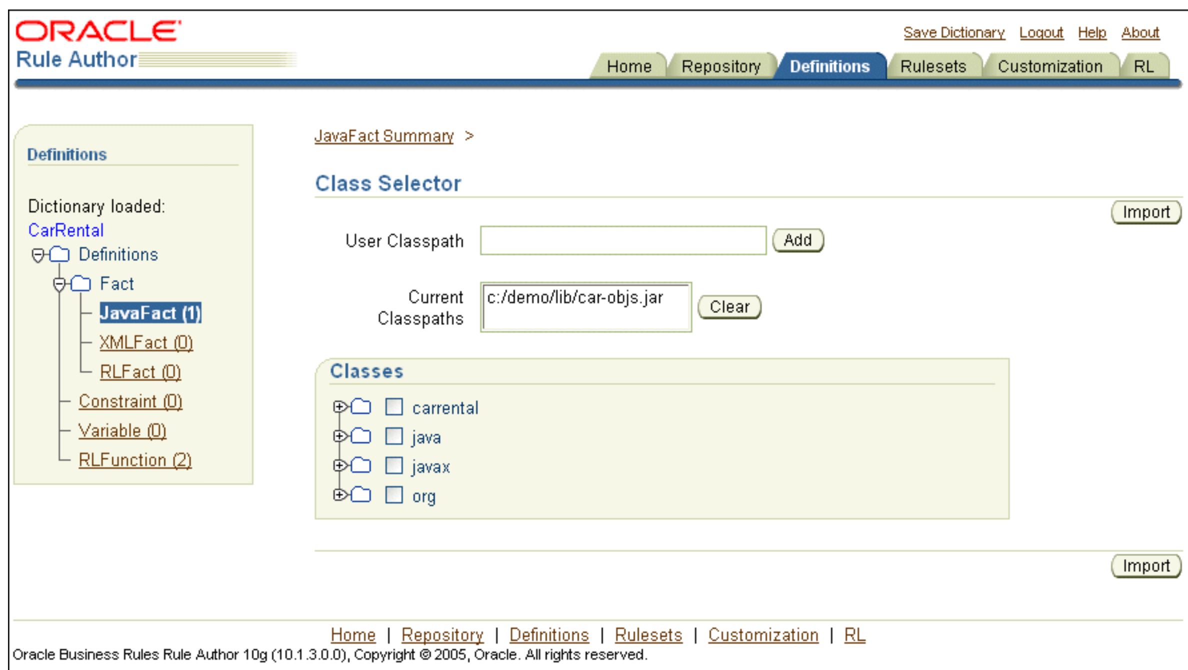



5. Click **Create**. This shows the Class Selector page.
6. On the Class Selector page, the **User Classpath** field lets you add a classpath. For example, for the car rental sample enter the following in the **User Classpath** field:

   $*HowToDir*`/lib/car-objs.jar`

   Where $*HowToDir* is the directory where you installed the Java Car Rental How-To.
7. Click **Add**. This updates the **Current Classpaths** field and adds the `carrental` package to the Classes box (see Figure 2–10).

***Figure 2–10 Rule Author JavaFact Class Selector Page***




**See Also:** Section 2.5.3 for more information on adding Java classes and packages to Rule Author.

## 2.5.3 Importing Java Classes to a Data Model

Next you need to select the desired Java classes to import into the data model from the Class Selector page. To add the `Driver` class to the data model, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. Click the **JavaFact** folder in the navigation tree.
3. Click **Create** on the JavaFact Summary page. This shows the Class Selector page.
4. In the Classes box on the Class Selector page, expand the carrental node and select the **Driver** check box (see Figure 2–11).
5. Click **Import**.

   Rule Author shows a confirmation message:

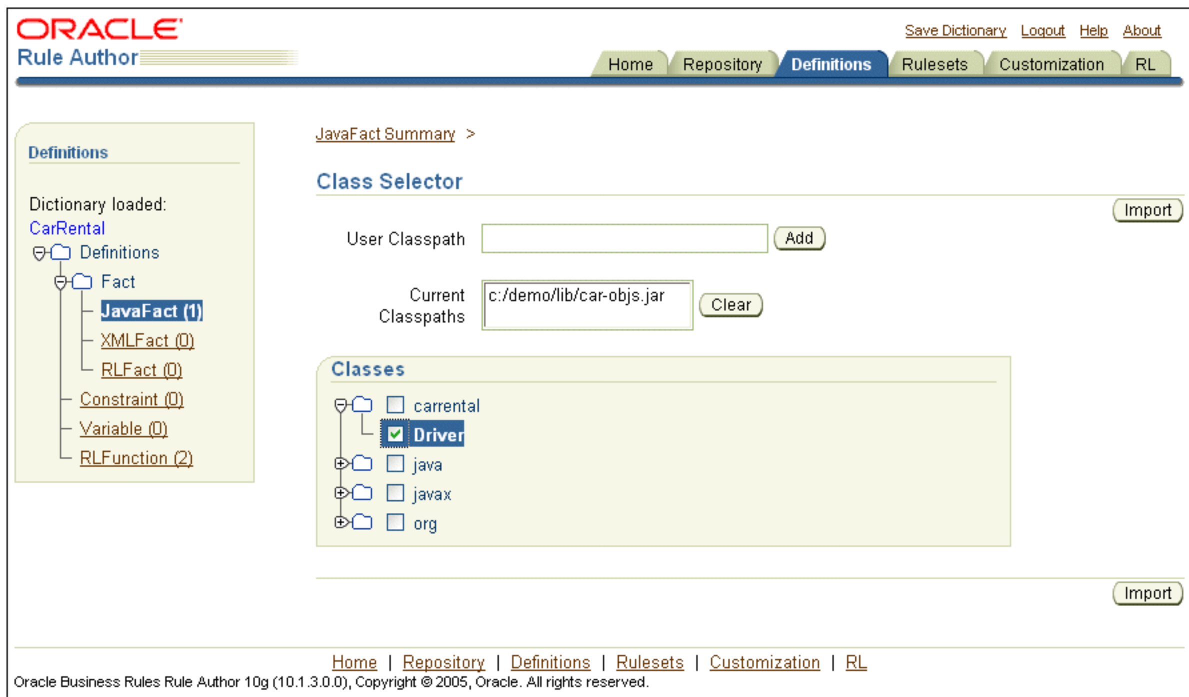
   ```
   1 class or package has been imported.
   ```

---

**Note:** After a class is imported the class selector page shows the class in bold.

---

***Figure 2–11 Rule Author JavaFact Class Selector Page***




Notes for specifying the user classpath and importing Java classes to Rule Author:

1. Using Rule Author, importing a Java Fact means the same thing as a Java import statement. That is, the classes and their methods become visible to Rule Author. Rule Author does not copy the Java code into the data model or into the dictionary.

2. The Class Selector page includes the Classes box that shows the Java classes available from the current classpaths.

3. The default Rule Author classpath includes three packages, `java`, `javax`, and `org`. These packages contain classes that Rule Author lets you import from the Java runtime library (rt.jar). Rule Author does not let you remove these classes from the Classes area (and the associated classpaths are not shown in the **Current Classpaths** field).

4. If you wish to assign or compare two objects that are not of the same type but are related by inheritance, you must import both classes to be compared and all classes between them in the inheritance hierarchy. For example, if you wish to assign an `ArrayList` to a variable of type `Object`, you must import `ArrayList`, `AbstractList`, and `AbstractCollection` into the data model. Otherwise, type-checking will not work correctly and expressions will not validate.

5. Classes and interfaces used in Rule Author must follow the following rules:

   a. If you are using a class or interface and its super class, the super class must be declared first.

   b. If you are using a class or interface, only its superclass or one of its implemented interfaces may be mentioned.

   For more information, see Section D.6, "Preserving Class Order and Hierarchies in the Data Model".

6. The Classes box navigation tree is rendered on demand (to improve performance). Thus, a child node is rendered only if its parent node is expanded. It is a good practice to keep only the nodes of interest expanded.
7. On Windows systems, you can use a "\" to specify the **User Classpath**. Rule Author accepts either path separator. For example, you can use the following: *$HowToDir*`\lib\car-objs.jar`.

   Where $*HowToDir* is the directory where you installed the How-To.
8. In the **User Classpath** you can specify a JAR file, a ZIP file, or a full path for a directory.
9. When you specify a directory name for the **User Classpath**, the directory specifies the classpath that ends with the directory that contains the "root" package (the first package in the full package name). Thus, if the **User Classpath** specifies a directory, Rule Author looks in that tree for directory names matching the package name structure.

   For example, if you want to import a class `cool.example.Test1`, located in `c:\myprj\cool\example\Test1.class` to the data model, you should specify the **User Classpath** value, `c:\myprj`.
10. Do not use RL reserved words in Java package names. For more information, see Section D.8, "Using RL Reserved Words as Part of a Java Package Name".

### 2.5.4 Saving the Current State of Definitions

While working on the data model from the **Definitions** tab and when you complete your work you should save the dictionary.

To save your definitions to the dictionary, do the following:

1. Click the **Save Dictionary** link.
2. Click **Save** on the Save Dictionary page.
3. Click the **Definitions** tab to return to the definitions page.

## 2.6 Defining Business Vocabulary for the Car Rental Sample

The business vocabulary allows business analysts, working with Rule Author to create rules using familiar names rather than using a Java package name, class name, method name, or member variable name. You use the Rule Author aliases feature to specify the business vocabulary. In this step you only need to define the business vocabulary for the business objects that you expect to use in rules. In addition, you can use the Rule Author **Visible** box to specify the properties and methods that show up in Rule Author lists when you create rules from the **RuleSets** tab.

This section covers the following topics:

- Specifying the Business Vocabulary for Java Fact Definitions
- Specifying the Business Vocabulary for Functions
- Specifying the Visibility for Properties and Methods

### 2.6.1 Specifying the Business Vocabulary for Java Fact Definitions

To specify the business vocabulary for Java Fact definitions, do the following:

1. Click the **Definitions** tab to view the Definitions page.

2. Click the **JavaFact** node in the navigation tree to display the JavaFact Summary page. For the car rental sample this shows a table that includes the imported class `carrental.Driver`.
3. Click the edit icon to view the JavaFact Properties and Methods for `carrental.Driver`.
4. For the name, `carrental.Driver`, enter the alias, `DriverData` in the **Alias** field (this **Alias** field is at the top of the page, under the **Name** field).
5. For the `age` entry in the Properties table, specify the desired alias. For example, enter `DriverAge` in the **Alias** field.
6. For the name entry in the Properties table, specify the desired alias. For example, enter `DriverName` in the **Alias** field.
7. Click **OK** to save your changes and return to the JavaFact Summary page.

**Note:** Be sure to click either **OK** or **Apply** after making changes. If you do not, Rule Author does not save your changes.

**See Also:** "Viewing Java Objects in a Data Model" on page 3-8

### 2.6.2 Specifying the Business Vocabulary for Functions

To specify the business vocabulary for an RL Language function, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. Click RLFunction in the Definitions folder in the navigation tree to display the RLFunction Summary page. For the car rental sample, this shows a table that includes the functions, `DM.assertXPath` and `DM.println`.
3. For the `DM.println` function, click the edit icon to view details.
4. In the **Alias** field, under the **Name** field, enter an alias. For example, enter `PrintOutput` in the **Alias** field.

**Note:** There is also an Alias field in the Function Arguments table. For this example, do not change the alias field in the function argument area.

5. Click **OK** to save your changes and return to the RLFunction Summary page.

### 2.6.3 Specifying the Visibility for Properties and Methods

To specify whether properties or methods are visible in Rule Author lists, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. Click the **JavaFact** node in the navigation tree to display the JavaFact Summary page. For the car rental sample this shows a table that includes the imported class `carrental.Driver`.
3. Click the edit icon to view the JavaFact Properties and Methods for `carrental.Driver`.

4. For each entry in the Properties table, specify the desired visibility using the Visible checkbox. For this example, only the member variables `age` and `name` need to be visible.
5. Click **OK** to save your changes and return to the JavaFact Summary page.

---

**Note:** Modifying the visibility indicators for a particular property or method may cause dependent definitions or rules to display incorrectly. If this occurs, mark any non-visible properties or methods causing the problem as visible.

---

## 2.7 Defining a Rule for the Car Rental Sample

In this section you define a rule for the car rental sample and see the basic steps for creating rules with Rule Author.

This section covers the following topics:

- Creating a Rule Set for the Car Rental Sample
- Creating a Rule for the Car Rental Sample

### 2.7.1 Creating a Rule Set for the Car Rental Sample

Before you can create a rule using Rule Author, you need to create a rule set. A rule set is a container for rules.

To create a rule set, do the following:

1. Click the **Rulesets** tab.
2. Click the **RuleSet** node in the navigation tree.
3. On the Ruleset Summary page, click **Create**. This displays the Ruleset page.
4. Enter text in the **Name** field. For example, enter `vehicleRent` in the **Name** field.

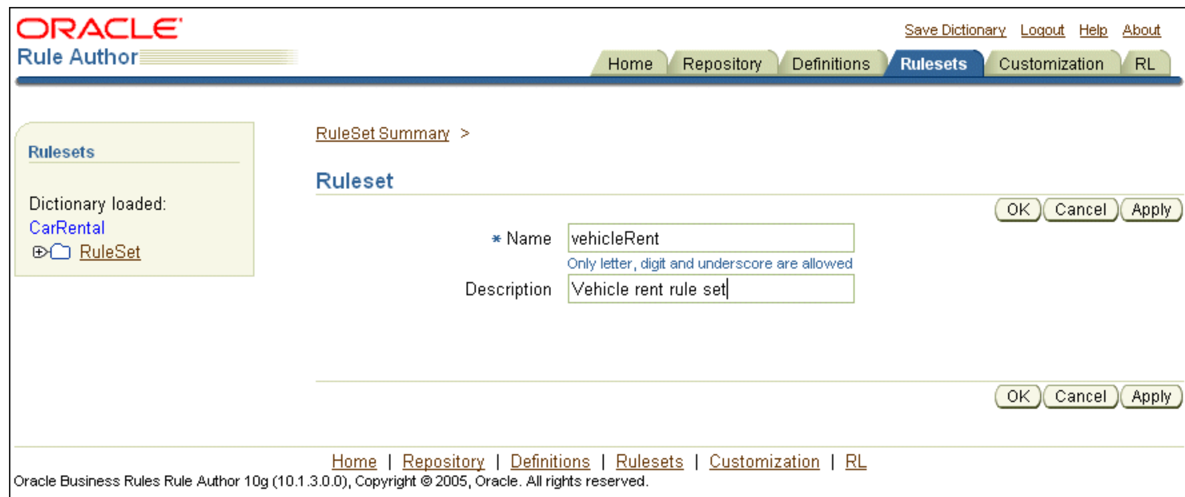
---

**Note:** Rule Author enforces a limitation for the name of a rule set; a rule set name can only contain letters (a-z and A-Z), numbers (0-9), and the underscore (_) character.

---

5. Optionally enter a description for the rule set in the **Description** field (see Figure 2–12).

*Figure 2–12 Rule Author Ruleset Page*




6. Click **OK** to create the `vehicleRent` rule set and exit the Ruleset page. After you click **OK**, the new rule set is visible in the navigation tree under RuleSet.
7. Save the dictionary.

---

**Note:** If you need to remove a rule set, do the following:

1. Select the RuleSet folder in the navigation pane.
2. Select the appropriate RuleSet in the RuleSet area by selecting the checkbox in the **Select** field.
3. Select **Delete**.

---

## 2.7.2 Creating a Rule for the Car Rental Sample

After creating a rule set you can create rules within the rule set. In this section, you will create a rule called `UnderAge`. This rule will test for the following:

If the driver's age is younger than 21, then decline to rent

The `UnderAge` rule contains a single pattern for the Rules Engine to match, and a test that is applied to the pattern.

The following actions are associated with the UnderAge rule:

- Print the text, "Rental declined", the name of the driver matched and the message, "Under Age, age is: " and the driver's age.
- Retract the matched driver object.

### 2.7.2.1 Adding the Under Age Rule for the Car Rental Sample

To use Rule Author to add the `UnderAge` rule, do the following:

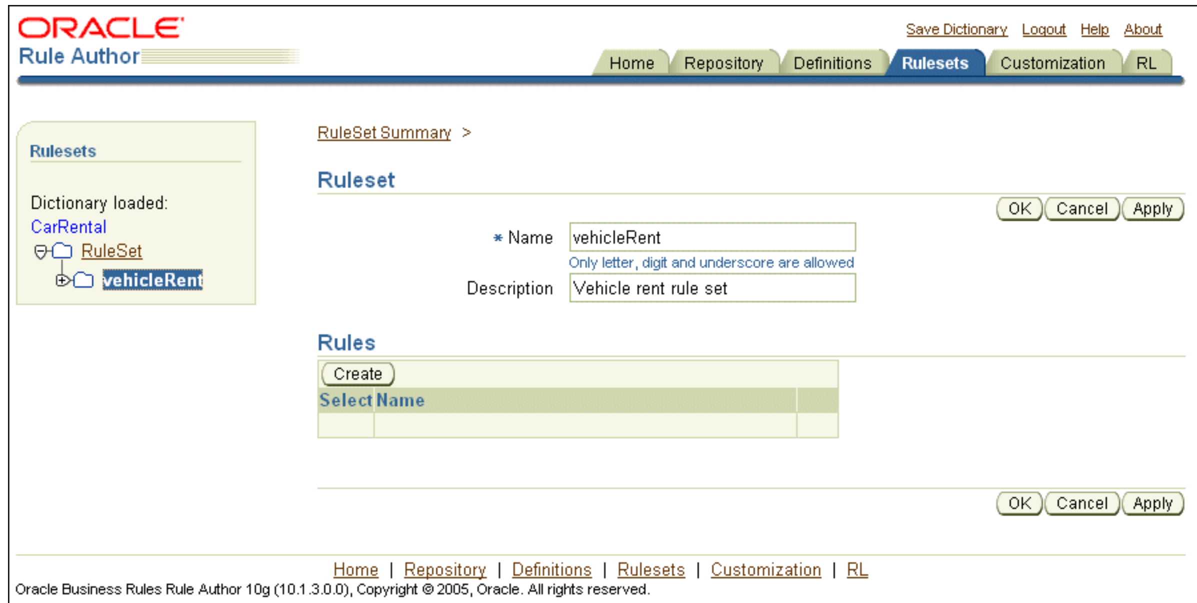
1. Click the **Rulesets** tab. The navigation pane displays the RuleSet folder that contains the `vehicleRent` rule set that you created in the section, "Creating a Rule Set for the Car Rental Sample" on page 2-15.
2. Click the `vehicleRent` folder in the tree. This displays the Ruleset page, with a table listing rules (see Figure 2–13).

---

**Note:** If this is the first rule you create then the Rules table is empty.

---

*Figure 2–13 Rule Author Ruleset Page Showing the Table of Rules*




3. Click **Create**. This displays the Rule page.
4. Enter `UnderAge` in the **Name** field.
5. Do not change the default value, 0, in the **Priority** field.

---

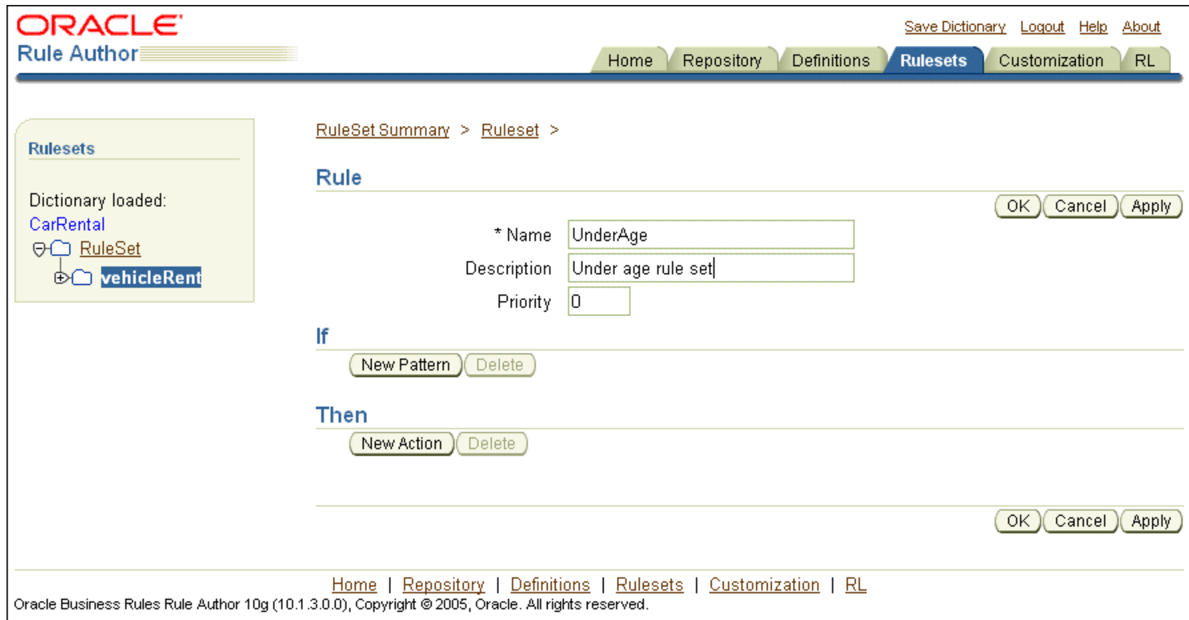
**Note:** The **Priority** field determines the rule priority. Rule priority specifies which rule to act upon, and in what order, if more than one rule applies within a rule set. Often in applications that use rules, the rules in a rule set are applied in any order until a decision is reached, and setting the rule priority is not required.

---

**See Also:** *Oracle Business Rules Language Reference Guide* for more information on working with rule priority.

6. Enter `Under age rule` in the **Description** field (see Figure 2–14).

*Figure 2–14 Rule Author Rule Page*




### 2.7.2.2 Adding a Pattern to the UnderAge Rule

When the Rules Engine runs it checks the facts against rule patterns to find matching patterns. You need to add a pattern for the `UnderAge` rule. Do the following to add a pattern to a rule:

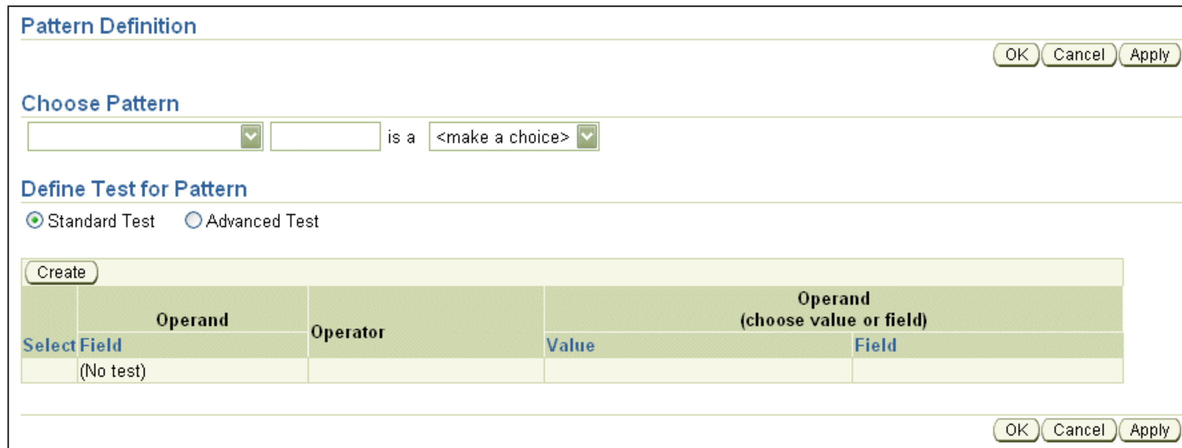
1. Click **New Pattern** in the **If** box on the Rule page. This brings up the Pattern Definition page which contains two areas: Choose Pattern and Define Test(s) for Pattern (see Figure 2–15).

---

**Note:** If the Pattern Definition page does not appear, you may have popup blocking enabled on your browser. Popup blocking must be disabled in order to use Rule Author.

---

*Figure 2–15 Rule Author Pattern Definition Page*



2. Under Choose Pattern, in the first box select the first choice (this shows no value in the selection box)

   This box specifies that the rule should fire each time there is a match (for all matching drivers). One alternate value, **There is at least one case**, selects one firing of the rule if there is at least one match (one such driver). The value **There is no case** specifies the rule fires once if there are no such matches (no matching drivers).

3. The next text area under Choose Pattern lets you enter a temporary name for the matched fact. Enter `driver` in this field (this defines the "pattern bind variable name").

   This value lets you test multiple instances of the same type in a single rule. For example, the pattern bind variable lets you compare a match for a driver with other drivers, using the specified name, in a comparison such as `driver1.age > driver2.age`.

4. The third box contains the text, `<make a choice>`. This box shows the available fact types from the data model. In this box select `DriverData` (if you did not define an alias for this in the data model, then this is `carrental.Driver)`.

5. Click **OK** or to save the pattern definition and return to the Rule page.

6. Click **OK** on the Rule page to save the rule.

   > **Note:** Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you navigate to a different rule set or select a different tab before you click **OK** or **Apply**, Rule Author discards the pattern definition.

7. Save the dictionary.

Without any tests defined on the pattern, the action which you define would apply to all drivers. To define tests for patterns, continue on the Pattern Definition page, as shown in, Section 2.7.2.3.

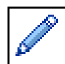
**See Also:** "Adding Actions for the Under Age Rule" on page 2-21

#### 2.7.2.3 Defining Tests for Patterns with the Under Age Rule

To add a test for a pattern, do the following:

1. From the **Rulesets** tab, in the navigation pane click the rule that you want to add a test. For this example, click the `UnderAge` rule in the navigation pane.

2. In the **If** box, click the pencil icon to display the Pattern Definition page. The Pattern Definition page contains two areas: **Choose Pattern** and **Define Test(s) for Pattern**.



3. On the Pattern Definition page, select the **Standard Test** button, then click **Create** (see Figure 2–16).

***Figure 2–16 Rule Author Rule Pattern Definition Page with Define Tests for Pattern Fields***

Standard pattern testing is only valid for AND expressions. Additionally, no grouping is allowed, and functions with parameters are not allowed. However, the use of constraints is allowed for customization. Advanced pattern testing does not have the restrictions of standard pattern testing, but the use of constraints is not allowed. Advanced expressions are not directly RL Language because aliases are used instead of variable names.

For more information, see Section 3.7.1, "Using the Advanced Test Expression Option".

4. In the **Operand** column, from the **Field** box, select `driver.DriverAge` (if you did not define an alias for this member variable in the data model, then this is `driver.age`).
5. In the **Operator** column, select < (less than).
6. In the **Operand** column, in the **Value** box enter 21. Do not enter a value in the **Field** box.
7. Next to the **Value** and **Field** boxes is a drop-down list containing the fixed values `Any` and `Fixed` (see Figure 2–17).

   These values are called constraints, and they are used to enable or disable customization for this field. Use the value `Fixed` to make the field read-only, which specifies that no customization is allowed for this field. Select the value `Any` to specify that Rule Author should allow changes to the value. Setting a value of `Any` allows for rule customization (which supports modifications by non-technical users). You can also define constraints that allow you to limit the allowed values.

   Select `Any` as the constraint for the **Value** field.

***Figure 2–17 Rule Author Pattern Definition Page With Values For Under Age Rule***

Pattern Definition

OK Cancel Apply

Choose Pattern

driver is a DriverData

Define Test for Pattern

Standard Test Advanced Test

Delete | Create

Select All | Select None

| Select | Operand Field | Operator | Operand (choose value or field) Value | | Field | |
|---|---|---|---|---|---|---|
| ☐ | driver.DriverAge | < | 21 | Any | <make a choice> | Fixed |

OK Cancel Apply

8. Click **OK** or save your changes and return to the Rule page.
9. Click **OK** on the Rule page.
10. Save the dictionary.

---

**Note:** Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you navigate to a rule set or select a different tab before you click **OK** or **Apply**, Rule Author discards the pattern definition.

---

**See Also:**

- "Customizing Rules for the Car Rental Sample" on page 2-24
- "Working with Constraints" on page 3-2

### 2.7.2.4 Adding Actions for the Under Age Rule

Actions are associated with pattern matches. When a rule's "If" portion matches, the Rules Engine activates the "Then" portion and prepares to run the rule's action.

In this section, you add two actions for the UnderAge rule. The first action prints the result. The second action retracts the driver fact from the Rules Engine. You might want to retract a fact for a number of reasons, including:

- If you are done with the fact, and you want to remove it from the Rules Engine.
- If the action associated with the rule changes the state, so that the fact needs to be retracted to represent the current state of the Rules Engine.

To add the action that prints the result for a match of the UnderAge rule, do the following:

1. Click the **Rulesets** tab.
2. In the tree, click the `UnderAge` node under the vehicleRent folder.
3. Click **New Action** on the Rule page in the Then box. This displays the Add Action page (see Figure 2–18).

***Figure 2–18 Rule Author Add Action Page***

Add Action

OK Cancel Apply

Choose an action.

Action Type <make a choice>

Action Parameters

Define parameters associated with the chosen action.

OK Cancel Apply

4. Select the `Call` item from the **Action Type** box. This shows the Action Parameters box.
5. Choose `PrintOutput` from **function** box (if you did not define an alias for this function, then this is `DM.println`). This expands the Function Arguments box.
6. Enter a value in the **Argument Value** field under Expression (see Figure 2–19). For example:

```
"Rental declined" + driver.DriverName + " Under age,age is:" + driver.DriverAge
```

---

**Note 1:** Rule Author uses a Java like syntax for expressions. The RL Language defines the complete expression syntax.

---

---

**Note 2:** If you do not know the variable names to use in the expression, use the edit icon in the Wizard field to bring up the expression wizard. This presents the wizard page which provides more space to write expressions. This also provides an easier and more accurate way to enter variables, since the expression builder presents a variable selection box.

---

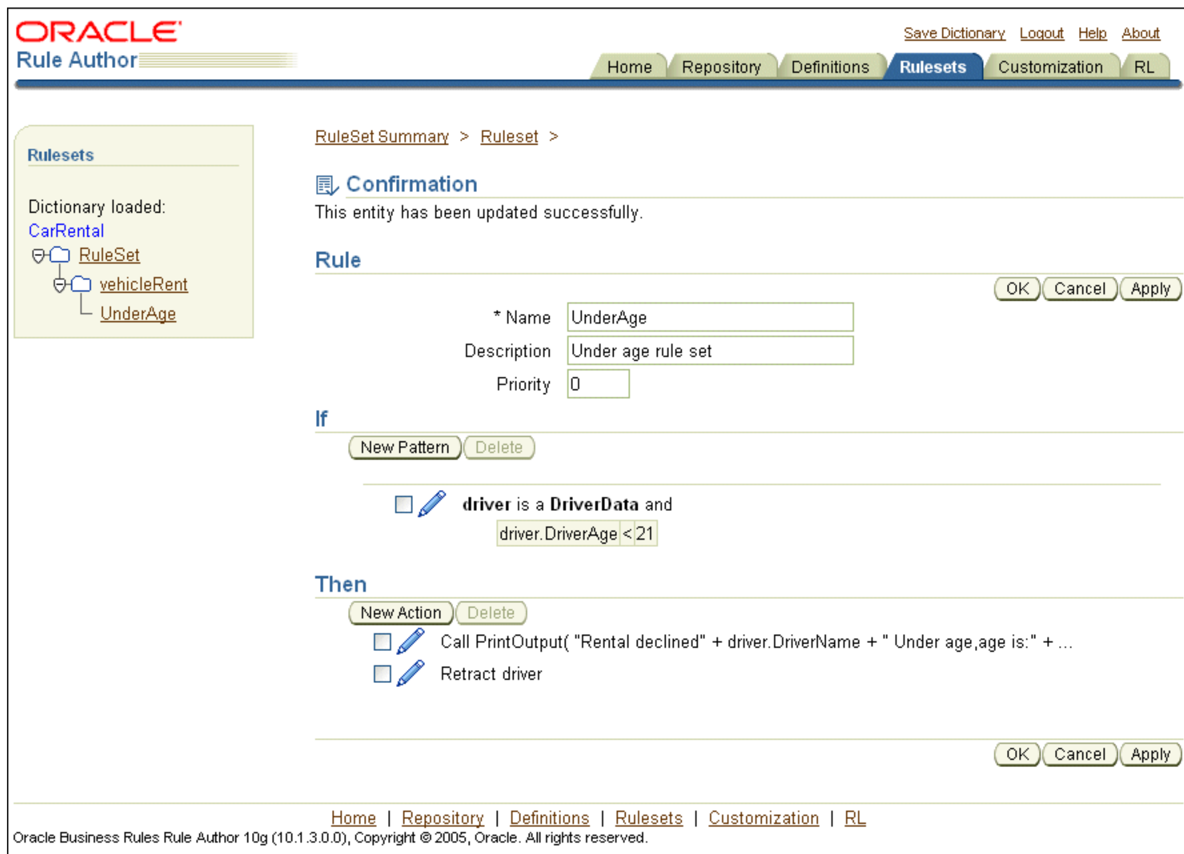
*Figure 2–19 Rule Author Add Action Page for Under Age Rule*

7. Click **OK** to save your changes and return to the Rule page.
8. Click **OK** on the Rule page.
9. Save the dictionary.

Next, add the retract action for the `UnderAge` rule. Perform the following steps to add this second action for the rule:

1. Click the **Rulesets** tab.
2. Click the `UnderAge` node under the vehicleRent folder.
3. On the Rule page, click **New Action** from the **Then** box. This brings up the Add Action page.
4. Select `Retract` from the **Action Type** box. This shows the **Action Parameters** box.
5. Select `driver` from the **Fact Instance** box. The pattern name `driver`, refers the single instance which was matched by the pattern.
6. Click **OK** to save your changes and return to the Rule page.
7. Click **Apply** on the Rule page to view the confirmation message (see Figure 2–20).
8. Save the dictionary.

*Figure 2–20 Rule Author Under Age Rule With Pattern and Actions*




---

**Note:** When you add actions to rules, you can only add new actions sequentially. If an action depends on the results of a previous action, then the order in which you add the actions is significant.

---

**See Also:** *Oracle Business Rules RL Language Reference Guide*

## 2.8 Customizing Rules for the Car Rental Sample

The Rule Author rule **Customization** tab is designed for business users. Rule developers use the **Allowed Values** field on the Pattern Definition page, which is available from the **Rulesets** tab, to specify if customization is allowed. When customization is allowed, you can specify a range of valid values for the customizable value. Then, business users may change values using the **Customization** tab.

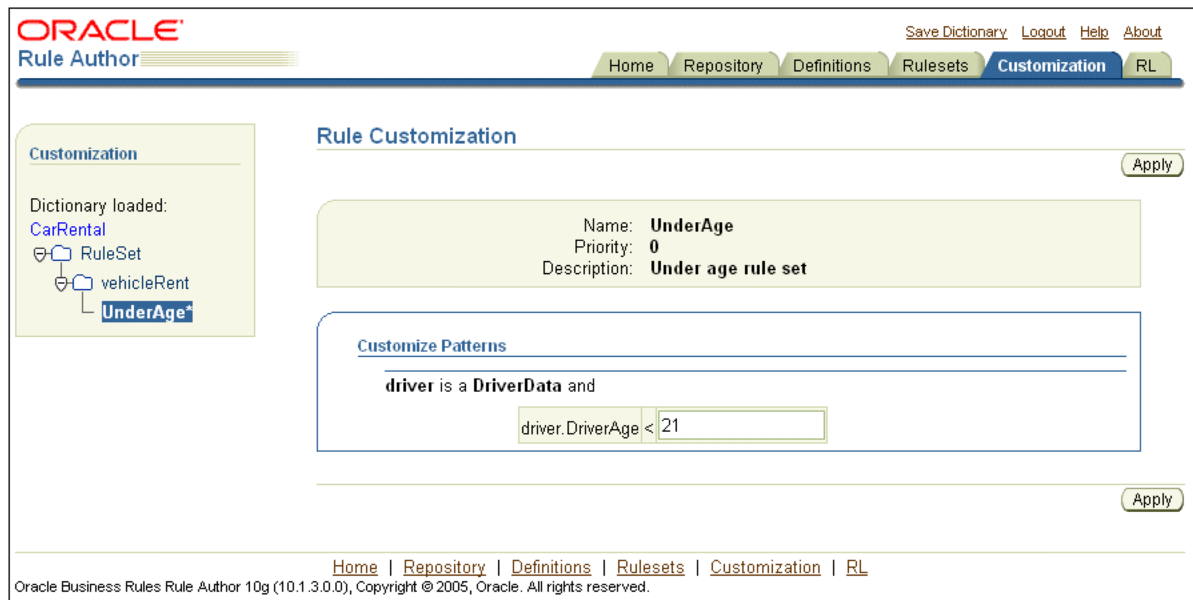
In this example, the `UnderAge` rule can be modified on the **Customization** tab to change the age of an under age driver (for this sample we do not limit values, and specify that any value is valid).

To change the `UnderAge` rule, use the **Customization** tab as follows:

1. Click the **Customization** tab. The navigation pane displays the vehicleRent folder with the `UnderAge` node followed by a "*", which indicates that the rule is customizable.

2. Click the node for the `UnderAge` rule (see Figure 2–21).

***Figure 2–21 Rule Author Rule Customization Page For Under Age Rule***




3. On the Rule Customization page the Customize Patterns box contains an editable text entry field for the test `driver.DriverAge < 21`.

   In this field, change the value 21 and enter the value 19 in this field.

4. Click **Apply**.
5. Save the dictionary.

After you save the dictionary, you are done creating the data model and the rules for the Java Fact How-To.

> **See Also:** "Defining Tests for Patterns with the Under Age Rule" on page 2-19 for information on the Allowed Values field.

## 2.9 Creating a Java Application Using Oracle Business Rules

After you create and save a dictionary that contains a data model and a rule set with rules, you can use the dictionary in a rule enabled Java application. This section shows you the steps for creating a rule enabled Java application.

> **Note:** Make sure your Java calls are wrapped in a try/catch block.

This section covers the following:

- Importing the Rules SDK and Rules RL Classes
- Initialize the Repository with Rules SDK
- Specifying a Rule Set and Generating RL with Rules SDK
- Initializing and Executing a Rule Session
- Asserting Business Objects Within a Rule Session
- Using the Run Function with a Rule Session

For the complete code for this sample application, see `TestMain.java` in the $*HowToDir*`/src/carrental` directory.

---

**Note:** The instructions in the preceding sections of this chapter enabled you to create and save a WebDAV repository and dictionary named `CarRental`. The car rental example supplied in the How-To sample code uses a file repository with a dictionary also named `CarRental`. The dictionary contents in the WebDAV repository you created in this chapter and the file repository in the How-To sample are identical.

The How-To sample code contains code for both WebDAV and file repositories, but only the file repository is described in detail. The How-To sample uses a file repository for portability, but this sample can be modified to use the WebDAV repository you created in the proceeding sections.

---

### 2.9.1 Importing the Rules SDK and Rules RL Classes

The first step when writing a rules enabled program is to import the required classes. Example 2–1 shows the imports from the `TestMain.java` application for the car rental sample.

***Example 2–1 Required Imports for Car Rental Sample With Rules SDK***

```
import java.util.Date;

import oracle.rules.sdk.ruleset.RuleSet;
import oracle.rules.sdk.repository.RuleRepository;
import oracle.rules.sdk.repository.RepositoryManager;
import oracle.rules.sdk.repository.RepositoryType;
import oracle.rules.sdk.repository.RepositoryContext;
import oracle.rules.sdk.dictionary.RuleDictionary;
import oracle.rules.sdk.exception.RepositoryException;

import oracle.rules.rl.RuleSession;

import carrental.Driver;
```

### 2.9.2 Initialize the Repository with Rules SDK

When building a rule enabled Java application, do the following to access a dictionary and specify a rule set (as shown in Example 2–2):

1. Create a `String` that contains the path to the repository.
2. Use a Rules SDK `RuleType` object to hold the repository that you obtain from the `RepositoryManager.getRegisteredRepositoryType` method. The `jarstoreKey` parameter specifies the repository type. Example 2–2 shows a file type repository.
3. Create a repository instance using the repository manager method `createRuleRepositoryInstance`.
4. Define a RepositoryContext and set appropriate properties. For a file repository, this step specified the path to the repository, as shown with the `repoPath` parameter in Example 2–2.

5. Use the `init` method in the `RuleRepository` object `repo` to initialize the repository instance.

***Example 2–2 Loading A Dictionary With Rules SDK***

```
String fs = "/";
String repoPath = "dict" + fs + "CarRepository";
final String jarstoreKey = "oracle.rules.sdk.store.jar";

RepositoryType jarType =
     RepositoryManager.getRegisteredRepositoryType( jarstoreKey );

RuleRepository repo = RepositoryManager.createRuleRepositoryInstance( jarType );

//fill in initialization property values
RepositoryContext jarCtx = new RepositoryContext();
jarCtx.setProperty( oracle.rules.sdk.store.jar.Constants.I_PATH_BASE, repoPath );

//initialize the repository instance.
repo.init( jarCtx );
```

## 2.9.3 Loading a Dictionary with Rules SDK

When building a rule enabled Java application you need to load a dictionary, with a specified version. Use a `RuleDictionary` object to load a dictionary, as shown in Example 2–3, which loads the `CarRental` dictionary, with the `HowTo` version, into the object named `dict`. The `CarRental` dictionary must be available in the repository (the `CarRental` dictionary with the version name `HowTo` was created earlier using Rule Author).

***Example 2–3 Loading a Dictionary With Rules SDK***

```
RuleDictionary dict = repo.loadDictionary( "CarRental", "HowTo" );
```

If you want to load a WebDAV repository instead of a file repository as shown in Example 2–3, you should use `getWebDAVRepository`. An example of this is shown in `TestMain.java` in the $*HowToDir*`/src/carrental` directory.

## 2.9.4 Specifying a Rule Set and Generating RL with Rules SDK

After loading a dictionary, you need to specify a rule set and use the Rules SDK to generate an RL Language program. This step is required since a dictionary stores a data model and rules using an intermediate XML format. The Rules SDK provides methods to access rule sets and rules the associated data model from a dictionary. The Rules SDK performs the mapping for the selected rule set from the intermediate XML format to produce the RL Language program that runs in the Rules Engine.

Using Rule Author, each rule set includes two components, a data model which is global and applies for all the rule sets in a dictionary, and the set of rules associated with a rule set. Example 2–4 shows the code that generates RL Language for these two components.

***Example 2–4 Generating Oracle Business Rules Rule Language With Rules SDK***

```
//init a rule session
String rsname = "vehicleRent";
String dmrl = dict.dataModelRL();
String rsrl = dict.ruleSetRL( rsname );
```

### 2.9.5 Initializing and Executing a Rule Session

After you generate an RL Language program that includes rules and a data model, you are ready to work with a rule session. A rule session initializes the Rules Engine and maintains the state of the Rules Engine across a number of rule executions. A `RuleSession` object is the interface between the application and the Rules Engine.

Example 2–5 shows the code that creates a `RuleSession` object and executes the RL Language program.

The `executeRuleset()` executes an RL program passed as a String. This method tells the Rules Engine to interpret the specified RL Language program.

> **Note:** The order of the `executeRuleset()` calls is important. You need to execute the data model RL Language program before the rule set RL Language program. The data model contains global information that is required when the associated rule set executes.

***Example 2–5 Initializing and Executing a Rule Session With Rules SDK***

```
RuleSession session = new RuleSession();
session.executeRuleset( dmrl );
session.executeRuleset( rsrl );

session.callFunction( "reset" );
session.callFunction( "clearRulesetStack" );
session.callFunctionWithArgument( "pushRuleset", rsname );
```

After the data model and the rule set are loaded and the rule session is ready to run the specified rule set against the facts that you assert for the rule session.

### 2.9.6 Asserting Business Objects Within a Rule Session

Before running a rule session you need to assert facts. When you execute a data model in a rule session, you prepare the rule session for new facts to be asserted. To assert facts, you use the `session.callFunctionWithArgument()` method and the assert function supplying a fact as an argument.

Example 2–6 shows sample code that prepares `Driver` objects for the car rental sample, and asserts three facts.

***Example 2–6 Preparing Driver and Accident Records For Car Rental Sample***

```
// Date Function
static public Date getDate(String dateStr ) {
  Date result = null;
  try {
    java.text.SimpleDateFormat sdf =
      new java.text.SimpleDateFormat( "MM/DD/YYYY" );
    result = sdf.parse( dateStr );
  }
  catch( Throwable t) { t.printStackTrace(); }
  return result;
}

// Driver d1 record
Date d1LicIssueDate = getDate( "10/1/1969" );
Driver d1 = new Driver( "d111", "Dave", 50, "sports", "full",
                        d1LicIssueDate, 0, 1, true );
```

```
// Driver d2 record
Date d2LicIssueDate = getDate( "8/1/2004" );
Driver d2 = new Driver( "d222", "Qun", 15, "truck", "provisional",
                        d2LicIssueDate, 0, 0, true );

//Driver d3 record
Date d3LicIssueDate = getDate( "6/1/2004" );
Driver d3 = new Driver( "d333", "Lance", 44, "motorcycle", "full",
                        d3LicIssueDate, 0, 1, true );

session.callFunctionWithArgument( "assert", d1 );
session.callFunctionWithArgument( "assert", d2 );
session.callFunctionWithArgument( "assert", d3 );
```

### 2.9.7 Using the Run Function with a Rule Session

Example 2–7 shows the code that runs a rule session.

***Example 2–7 Running A Rule Session with the Run Function***

```
session.callFunction( "run");
```

## 2.10 Running the Car Rental Sample Using the Test Program

The *$HowToDir*/`lib` directory includes `TestMain.jar`, a ready-to-run Oracle Business Rules Java application that uses the CarRental dictionary. If you change the dictionary name then you need to modify `TestMain.java` (the source is available in the directory *$HowToDir*/`src`).

Example 2–8 shows output from running `TestMain` using the facts asserted within `TestMain`.

---

**Note:** The `Readme.txt` file in the *$HowToDir*/`src` directory includes instructions for setting the environment variables required to run the test program.

---

***Example 2–8 Sample Run Of Car Rental Program***

```
java carrental.TestMain
Rental declined Qun Under age: age is: 15
```

Note that not all facts produce output or fire a rule. The example shows the output only for the asserted fact that matches the UnderAge rule.

# 3 Working With Rule Author Features

This chapter describes how to use some of the more advanced features of Oracle Business Rules Rule Author. The following topics are covered:


- Working with Variables
- Working with Constraints
- Working with RLFacts
- Working with Functions
- Viewing Java Objects in a Data Model
- Generating Oracle Business Rules RL Language Text
- Configuring Rule Author Dictionary Properties
- Deleting a Rule Author Dictionary
- Importing and Exporting a Dictionary
- Working with Test Rulesets
- Invoking Rules


## 3.1 Working with Variables

In this section you use Rule Author to add a variable that replaces a portion of the message that you print out in the Java Car Rental How-To you built in Chapter 2. Using Oracle Business Rules, a variable is similar to a public static variable in Java. You can specify that a variable is a constant or modifiable.

Perform the following steps to add a variable:

1. Click the **Repository** tab and load the CarRental dictionary.
2. Click the **Definitions** tab.
3. Click the **Variable** node in the navigation tree. The Variable Summary page shows that no variables are defined.
4. Click **Create**. This shows the Variable page.
5. Enter `DeclineMessage` in the **Name** field.

> **Note:** When Rule Author creates a variable, it adds a "`DM.`" to the name you enter in the **Name** field (`DM` stands for Data Model).

6. Enter `Decline Message` in the **Alias** field.

7. Select the **Final** check box (by default this box is selected).
8. Select `String` in the **Type** box.
9. In the **Expression** box, enter `"Rental declined "`.

   If you want to use the expression wizard to assist you with creating an expression, click the edit icon to bring up the expression builder.
10. Click **Apply** (see Figure 3–1).

***Figure 3–1 Rule Author Variable Definition Page***



Notes for creating Rule Author variables:

- When you deselect the **Final** check box, this specifies that the variable is modifiable, for instance, in an Assign action.
- You can only use variables specified as final variables in the test for a rule (non-finals are not allowed).

**See Also:** "Defining Tests for Patterns with the Under Age Rule" on page 2-19 for an example of a test for a rule.

## 3.2 Working with Constraints

When you want to constrain the allowed values for a field to only a specific set of values in a customizable rule, for example if you want to specify a range of values, you can use a Rule Author constraint definition.

Rule Author supports three types of constraint definitions, as shown in Table 3–1.

*Table 3–1 Rule Author Constraint Types*

| Constraint Type | Description |
|---|---|
| Range | Specifies a numeric interval. |
| Enumeration | Specifies a list of possible values. |
| Regular Expression | Specifies a regular expression to which the string value conforms. The syntax for the regular expression in these constraints follows Java's regular expression definition. |

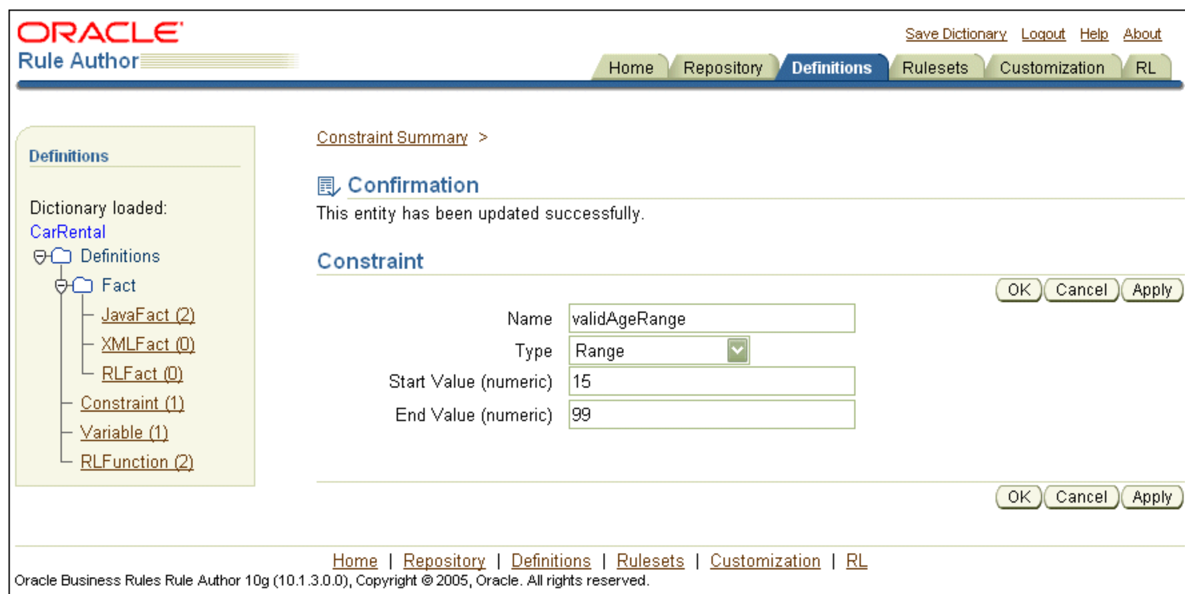
**Note:** The regular expression constraint requires quotation marks around strings.

For the example in this section you define a constraint and then add the constraint to the UnderAge rule in the CarRental dictionary.

Perform the following steps to define a range constraint:

1. Click the **Repository** tab and load the CarRental dictionary.
2. Click the **Definitions** tab.
3. Click the **Constraint** node in the navigation tree. The Constraint Summary page shows no constraints are defined.
4. Click **Create**. This shows the Constraint page.
5. Enter `validAgeRange` in the **Name** field.
6. Select `Range` from the **Type** box. This shows a Constraint page with two new fields: **Start Value** and **End Value**.
7. Enter 15 in the **Start Value** field.
8. Enter 99 in the **End Value** field.
9. Click **Apply**. Rule Author shows a confirmation message (see Figure 3–2).

***Figure 3–2 Rule Author Constraint Definition Page***




10. Save the dictionary.

Next, add the `validAgeRange` constraint to the UnderAge rule. To use the constraint in the UnderAge rule, do the following:

1. Click the **Repository** tab and load the CarRental dictionary.
2. Using the CarRental dictionary, select the **Rulesets** tab.
3. Select the UnderAge link to show the UnderAge rule.
4. Select the edit icon in the **If** box. This displays the Pattern Definition page.
5. On the Pattern Definition Page, in the constraint field, select `validAgeRange` (this is the second box in the Value column).
6. Click **OK**. This closes the Pattern Definition Page.
7. Click **OK** on the Rule page.
8. Save the dictionary.

Use the **Customization** tab to verify that Rule Author lets you enter values from the specified range and rejects invalid entries.

> **See Also:** "Defining Tests for Patterns with the Under Age Rule" on page 2-19 for information on using the Allowed Values field to use constraints.

---

**Note:** If you change a constraint that is used in a ruleset, you can still save the ruleset even though it may not adhere to all the constraints.

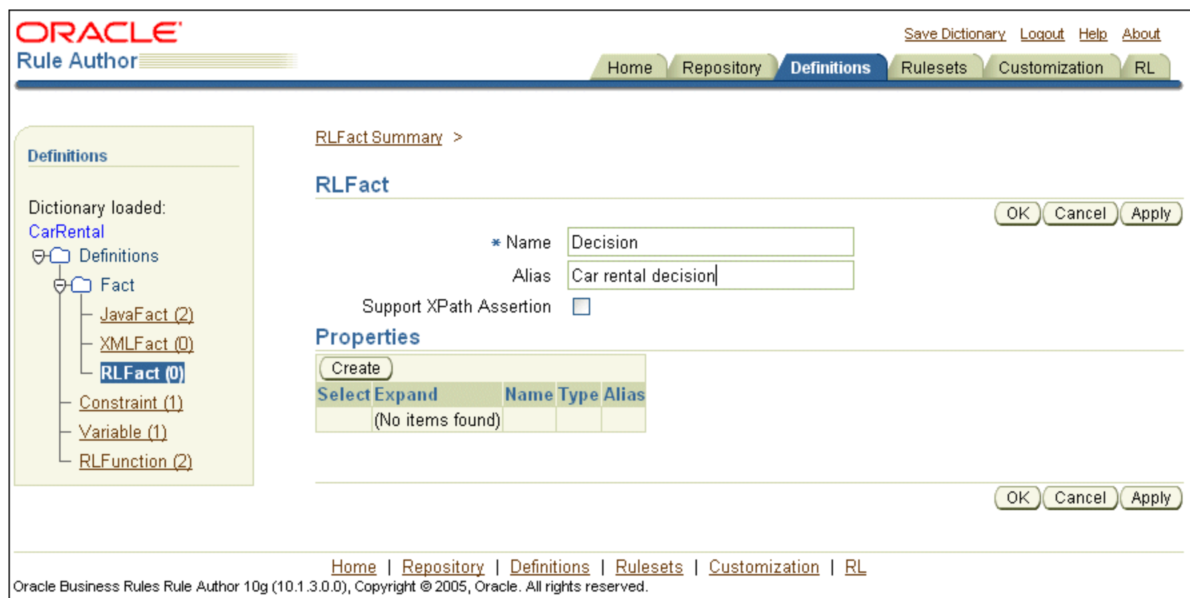
---

## 3.3 Working with RLFacts

This example creates an RLFact named Decision that extends the CarRental rules. The RLFact has three members of String type: driverName, type, and message. Perform the following steps to create the Decision RLFact:

1. Click the **Repository** tab and load the CarRental dictionary.
2. Click the **Definitions** tab and click the **RLFact** node in the navigation tree under Facts. The RLFact Summary page shows that no RLFacts are defined.
3. Click **Create**. This shows the RLFact page.
4. Enter `Decision` in the **Name** field.
5. Enter `Car rental decision` in the Alias field (see Figure 3–3).

***Figure 3–3 Rule Author Definitions Tab with RLFact Page***




6. In the properties table click **Create**. This shows a new row in the Properties table.
7. Enter `driverName` in the **Name** field.
8. Select `String` from the box in the **Type** field.
9. Enter `driver name` in the **Alias** field.
10. Click **Create**. This adds another new row to the Properties table.
11. Enter `type` in the **Name** field.
12. Select `String` from the box in the **Type** field.
13. Enter `decision type` in the **Alias** field.
14. Click **Create**. This adds another new row to the Properties table.
15. Enter `message` in the **Name** field.
16. Select `String` from the box in the **Type** field.
17. Enter `message for decision` in the **Alias** field.

18. Click **Apply**. This displays a confirmation message (see Figure 3–4).

***Figure 3–4 Rule Author Definitions Tab with RLFact Properties***

19. Click RLFact in the navigation tree. This displays the RLFact Summary page, and the new RLFact, `DM.Decision`.

---

**Note:** When Rule Author creates an RLFact, it adds a "`DM.`" to the name you enter in the **Name** field (the `DM` stands for Data Model).

---

**See Also:** "Specifying Visibility and Object Chaining for Rule Author Drop Down Lists" on page 3-9 for information on the **Expand** field shown in Figure 3–4.

## 3.4 Working with Functions

Oracle Business Rules lets you use built-in or user-defined functions in rule conditions and actions. In this section you use Rule Author to define a function named `showDecision`. You can use this function to print the results for the Java How-To.

---

**Note 1:** The example in this section uses the CarRental dictionary and the RLFact defined in Section 3.3.

---

---

**Note 2:** For RL generated from the SDK (for example, Rule Author), global variables may not be referred to directly in an RL function. For more information, see Section D.2, "Global Variables may not be Used in RL Functions".

---

Do the following to define the `showDecision` function:

1. Click the **Repository** tab and load the CarRental dictionary.
2. Using the CarRental dictionary, select the **Definitions** tab.
3. Select **RLFunction** in the navigation tree. This shows the RLFunction Summary page.
4. Click **Create**.
5. Enter `showDecision` in the **Name** field.
6. Enter `Show Decision` in the **Alias** field.

---

**Note:** If you are defining a function in Rule Author, you must specify a valid alias in the **Alias** field, even though the actual function name (not the alias) must be used in the function body.

---

7. Select `void` in the box in the **Return Type** field (this is the default value).
8. Click **Create** in the **Function Arguments** table.
9. Enter `decision` in the **Name** field.
10. Enter `Decision made for driver` in the **Alias** field.
11. Select `Car rental decision`, the alias for the Decision RLFact, in the box in the **Type** field.
12. In the Function Body box, enter the following:

```
DM.println( "Rental decision is " + decision.type + " for driver " +
decision.driverName + " for reason " + decision.message);
```

13. Click **Apply**. This shows a confirmation message.
14. Click the RL Function node in the left navigation pane. You should see the RL function `DM.showDecision` in the summary table.
15. Click Edit to view the function (see Figure 3–5).

***Figure 3–5 Rule Author RLFunction Page***

After creating the new RLFact `Decision` as specified in Section 3.3 and the new RLFunction `DM.showDecision`, you can update the UnderAge rule to provide an action that creates a new Decision fact. To use the Decision fact with the `showDecision` function, you need to create a new rule that checks for Decision facts and provides an action, using the `showDecision` function, to show the results.

## 3.5 Viewing Java Objects in a Data Model

To view the objects in a data model, including any classes or packages that you import, do the following:

1. Click the **Repository** tab and load the appropriate dictionary. For example, load the CarRental dictionary.
2. Click the **Definitions** tab to view the Definitions page.
3. Expand the Facts folder and click the **JavaFact** node in the navigation tree to display the JavaFact Summary page.

   For the car rental sample this shows a table that includes the imported class `carrental.Driver`.

4. Click the edit icon to view the Java Fact Properties and Methods.

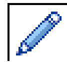


*Table 3–2 JavaFact Summary Fields*

| Field | Description |
|---|---|
| **Name** | The name of the Java Object. |
| **Alias** | The specified alias name for the Java Object that is shown in Rule Author lists. |
| **Visible** | This box specifies if the Java Object is shown in Rule Author lists. |
| **Support XPath Assertion** | This box implies you can use the class in XPath expressions to assert XML data into a rule session. |

*Table 3–3 JavaFact Properties and Methods Fields*

| Field | Description |
|---|---|
| **Visible** | Specifies that the property or method shows up in Rule Author lists. |
| **Expand** | Specifies that the superclass for a property or method that has a superclass shows up in Rule Author lists. |
| **Member Variable Name** | This is shown for Properties. Specifies the property name. |
| **Type** | Specifies the type for a property |
| **Alias** | This is a text field that you can modify to specify the business vocabulary for the property or object. The specified name is used when the object is shown in Rule Author drop down lists. |
| **Method Name** | This is shown for methods. |
| **Argument Type** | This is shown for methods. |
| **Return Type** | This is shown for methods. |

**Note:** Importing a Java class causes its super class and classes associated through fields and methods to be imported into the data model. The JavaFact Summary page table shows you the super class and the associated classes for any classes that you import.

### 3.5.1 Specifying Visibility and Object Chaining for Rule Author Drop Down Lists

You can specify that properties, classes, or methods are visible or not visible in Rule Author selection boxes (the selection boxes that contain classes, properties, and methods are shown when you create rules on the **Rulesets** Tab).

**Note:** For the Java Fact How-To, you do not need to change the object chaining.

To remove the visibility of a Java object, do the following:

1. At the top of the Java Fact page, use the **Visible** box to specify whether an object is visible (by default, objects are visible).

2. Deselect the Visible box to remove the object from Rule Author selection boxes.
3. Click **OK** on the Java Fact page.

To remove visibility of a Java property or method, do the following:

1. Deselect the a property or method in the in the Properties area or the Methods area on the Java Fact page to remove the property or method from Rule Author selection boxes.
2. Click **OK** on the Java Fact page.

You can specify that Rule Author selection boxes show the methods or properties one level above a specified method or property, in superclass chain, by selecting the **Expand** box for the method or property on the Java Fact page. The Expand box is shown in the **Expand** field of the Properties and Methods area. The **Expand** box is only shown when a method or property includes a superclass (Rule Author does not show the **Expand** box for primitive types).

## 3.6 Generating Oracle Business Rules RL Language Text

Using the **RL** tab, Rule Author lets you view the RL Language text that represents the data model and the rule sets associated with the dictionary data.

### 3.6.1 Generating and Viewing an RL Language Program

To generate and view RL Language text, do the following:

1. Click the **Repository** tab and load a dictionary. For example, load the CarRental dictionary.
2. Select the **RL** tab.
3. Select the rule set of interest in the navigation tree.
4. Click **Generate RL**. This shows you the RL Language text for the specified ruleset.
5. Click **Check RL Syntax** to validate the RL Language text.

## 3.7 Configuring Rule Author Dictionary Properties

Two properties can be configured globally for an entire dictionary in Rule Author:

- Advanced Test Expression
- Logging

To configure these properties, access the Dictionary Properties page:

1. Make sure you are connected to a repository and a dictionary is loaded.
2. Click the **Repository** tab.
3. Click the **Properties** secondary tab (see Figure 3–6).

*Figure 3–6 Rule Author Dictionary Properties Page*

### 3.7.1 Using the Advanced Test Expression Option

The **Advanced Test Expression** check box changes the Rule Author expression mode to advanced for test expressions displayed when you edit a pattern for a rule (see Figure 3–7). Tests in a rule's condition can involve mathematical operations and conjunctions. Rule Author includes the advanced expression mode to support defining such complex expressions.

When a pattern is first created, its test expression mode depends on the **Advanced Test Expression** property set on the Properties page. When you select the Advanced Test Expression check box, then the advanced test expression mode is applied to all new patterns. This setting persists when the dictionary is saved. In this case, when the dictionary is loaded, all patterns are created with Advanced mode. After a pattern is created, with or without a test, it is permanently associated with the test expression mode. Thus, the Test Expression mode of a pattern cannot be changed.

In a single rule, you can use both patterns created with basic expression mode and advanced expression mode.

***Figure 3–7 Pattern Definition Advanced Test Expression Page***

### 3.7.2 Using the Logging Option

The **Logging** box specifies the logging options. This option is useful when you need to report a problem with Rule Author. To specify logging select the **Logging** check box and then select the following log file properties:

- **Log Level**: **Error** is the lowest log level and generates the least amount of output, **Status** is the medium log level, and **Trace** is the highest log level, generating the most amount of output.
- Use the **Log Directory** box to specify the directory for the log. Specifying the log directory in the **Log Directory** box is optional (if the directory is not set, the log is displayed on the console).
- When the **Log file name** is specified, the log is saved in a file with the name *<log_file_name>.<last_8_session_id>*. For example, if you specified RALog as the name of your log file, and the last eight digits of your session ID are 11223344, the file RALog.11223344 would be created. If no log file is specified, the log is saved in a file named RuleAuthor.*<last_8_session_id>*.

Click **Update** when you are finished specifying your logging options.

## 3.8 Deleting a Rule Author Dictionary

This section shows you how to delete a version in a dictionary or delete an entire dictionary.

If you want to delete an individual dictionary version, do the following:

1. Click the **Repository** tab.
2. Click the **Delete** secondary tab.

   If you want to delete a specific dictionary version, select a dictionary and version in the "Select dictionary version" section, then click **Delete Version**.

   If you want to delete an entire dictionary (and all of its versions), select the dictionary in the "Select entire dictionary" section, then click **Delete**.

## 3.9 Importing and Exporting a Dictionary

You can import a specific version of a dictionary or an entire dictionary into Rule Author. To do so:

1. Click the **Repository** tab.
2. Click the **Import** secondary tab.

   If the dictionary you want to import resides locally, use the section in the first bullet to specify its location. You can manually enter the path to the dictionary or click the Browse button and select the dictionary.

   If the dictionary you want to import resides on a different machine (where the Rule Author is running), you must specify the full path to the dictionary on that server.
3. Click **Import**.

To export an entire dictionary:

1. Click the **Repository** tab.
2. Click the **Export** secondary tab.
3. In the "Select entire dictionary" section:
   a. In the Dictionary field, select the dictionary you want to export.
   b. In the File Location field, specify the location and filename (absolute file path on the server) to which you want to export the dictionary.
4. Click **Export**.

   You can also select the dictionary and click **Download**, which creates a link to the exported archive on the Export Dictionary page. You can then click the link and use your browser to download the archive to a location of your choice (see Figure 3–8).

*Figure 3–8 Rule Author Export Dictionary Page Showing the Download Option*

To export a specific dictionary version:

1. Click the **Repository** tab.
2. Click the **Export** secondary tab.
3. In the "Select dictionary version" section:
   a. In the Dictionary field, select the dictionary you want to export.
   b. In the Version field, select the dictionary version you want to export.
   c. In the File Location field, specify the location and filename (absolute file path on the server) to which you want to export the dictionary.
4. Click **Export**.

   You can also select the dictionary and version and click **Download**, which creates a link to the exported archive on the Export Dictionary page. You can then click the link and use your browser to download the archive to a location of your choice (see Figure 3–8).

## 3.10 Working with Test Rulesets

The Test Rulesets feature in the Rule Author is designed to allow you to write RL functions that test the rule sets created in Rule Author. The selected rule set and its associated data model are inserted into a Rule Session where a user-defined function is called which uses the rule set.

To use the Test Rulesets feature in Rule Author:

1. Create a rule set you want to test. If the data model includes any Java classes, the Java classes must be included in the OC4J classpath. The easiest way to do this is to put the JAR files in the following directory, then restart OC4J:

   ```
   $ORACLE_HOME/j2ee/home/applications/ruleauthor/lib
   ```

   You can also include the Java classes as a shared library. This allows Rule Author to share the classes with other applications. To do this, login to Enterprise Manager and perform the following:

   a. Navigate to the OC4J:home page.
   b. Click the **Administration** tab.
   c. Find the Shared Libraries node under the Properties node and click the icon in the Go to Task column.
   d. Click **Create**, enter the library name and version number, then click **Next**.
   e. Click **Add**, navigate to the location of the JAR file, then click **Continue**.
   f. Click **Finish** to return to the Shared Libraries page.
   g. Find and click the link to the library you just created. Copy the absolute path to the archive.
   h. Return to the OC4J:home page.
   i. Click the **Applications** tab.
   j. Click the link to the Rule Author application (this was defined when you first deployed the Rule Author application).
   k. Click the **ruleauthor** module link.
   l. Click the **Administration** tab.
   m. Find the Configuration Properties node and click the Go to Task icon.
   n. In the Classpath field, paste the absolute path to the archive which you copied in Step g, then click **OK**.
   o. In the resulting confirmation message, click the **Restart** link to restart Rule Author.

2. Create a function that has some print statements to indicate the execution of the function. In this example, a function called `DM.test` is created:

   a. Click the **Definitions** tab.
   b. Click the **RLFunctions** node in the navigation tree.
   c. Enter `DM.test` in the Name and Alias Fields.
   d. Leave `void` as the return type.
   e. Enter the following in the Function Body field:

   ```
   java.text.SimpleDateFormat sdf =
      new java.text.SimpleDateFormat( "MM/dd/yyyy" );

   assert (new carrental.Driver( "d111", "Dave", 50, "sports", "full",
                                 sdf.parse ("10/1/1969"), 0, 1, true ));
   assert (new carrental.Driver( "d222", "Abe", 15, "truck", "provisional",
                                 sdf.parse ("8/1/2004"), 0, 0, true ));
   assert (new carrental.Driver( "d333", "Lance", 44, "motorcycle", "full",
                                 sdf.parse ("6/1/2004"), 0, 1, true ));
   ```

```
pushRuleset ("vehicleRent");
run();
```

This function asserts several facts, pushes the `vehicleRent` rule set onto the rule set stack, and calls `run()`.

---

**Note:** All rule sets you want to have executed must be pushed onto the stack. They are not automatically pushed simply by selecting them.

---

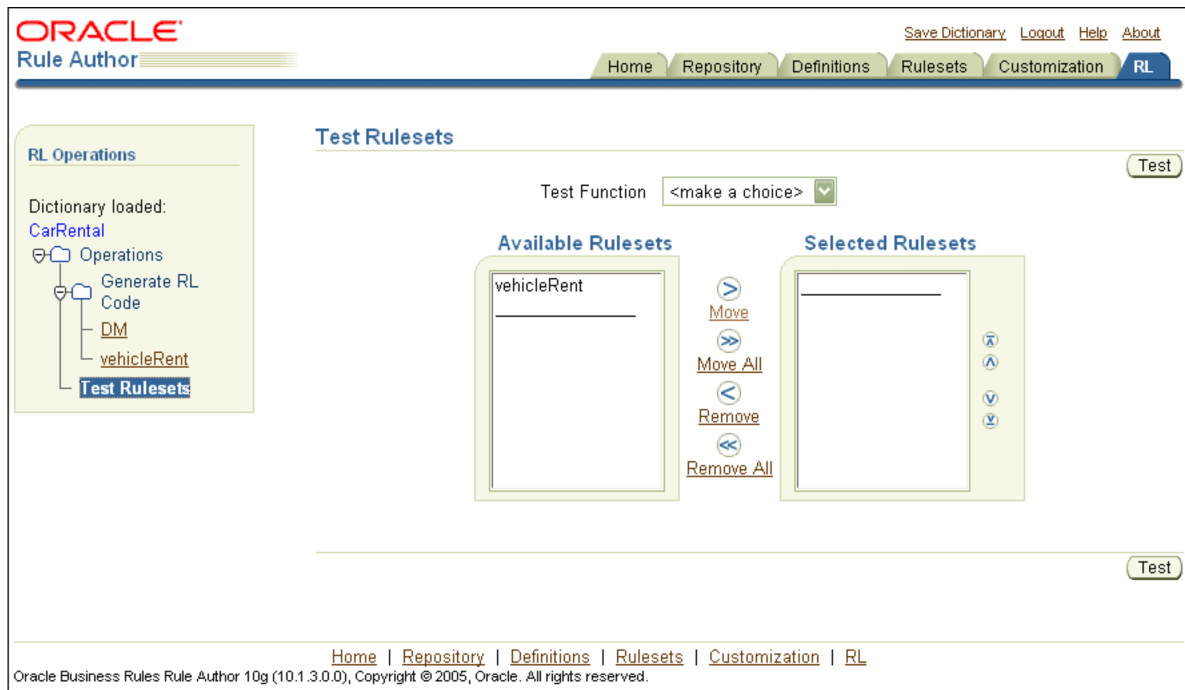
If you wanted to enable logging, you could call the `watchFacts()` and `watchActivations()` functions.

f. Click **OK**.

g. Save the dictionary.

3. Click the **RL** tab.
4. Click the **Test Rulesets** node in the navigation tree. This displays the Test Rulesets page.

***Figure 3–9 Rule Author Test Rulesets Page***

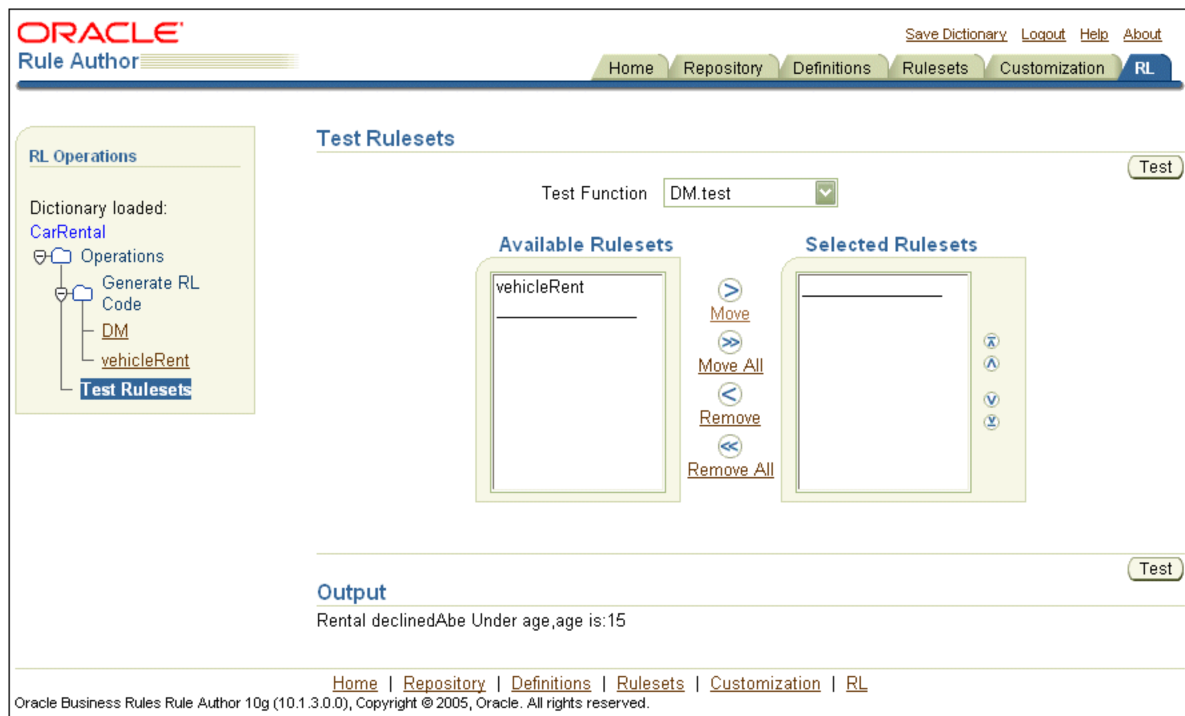



You can see the `vehicleRent` rule set is available. This rule set was created in Chapter 2.

5. Select the `vehicleRent` rule set and move it to the Selected Rulesets column.
6. Select the **DM.test** function from the Test Function list.
7. Click **Test**.

RL code is generated for the selected rule set(s), which is then inserted into a Rule Session. Then, the selected test function is called. Output from the function is printed on the screen (see Figure 3–10).

***Figure 3–10 Rule Author Test Rulesets Page with Output***




## 3.11 Invoking Rules

A rule enabled program usually invokes rules with the following steps:

1. Pass objects to the rule engine to be asserted as facts.
2. Run rules.
3. Obtain results produced from rules that fired.

This section describes the best practices for using an RL function to encapsulate invoking rules. This section covers three techniques for invoking rules; the techniques differ primarily in the details required to obtain results. This section uses a sample RL function named `getSubscribers`. Using this routine, each approach for invoking rules looks identical from the point of view of the rule enabled program that calls the `getSubscribers`.

This section covers the following:

- Overview of Results Examples
- Using a Global Variable to Obtain Results
- Using Container Objects to Obtain Results
- Using Reasoned On Objects to Obtain Results

### 3.11.1 Overview of Results Examples

The examples in this section show a highway incident notification system. These examples show the different approaches to access the results of rule engine evaluation. The examples use two Java classes: `traffic.TrafficIncident` and `traffic.IncidentSubscription`.

> **Note:** The `traffic.*` sample classes are not included in the Oracle Business Rules distribution.

The `TrafficIncident` class represents information about an incident affecting traffic and contains the following properties:

- Which highway
- Which direction
- Type of incident
- Time incident occurred
- Estimated delay in minutes

The `IncidentSubscription` class describes a subscription to notifications for incidents on a particular highway and contains the following properties:

- Subscriber - the name of the subscriber
- The highway
- The direction

In the example using these classes, when an incident occurs that affects traffic on a highway, a `TrafficIncident` object is asserted and rule evaluation determines to whom notifications are sent.

In the examples, the `sess` object is a `RuleSession` and a number of incident subscriptions are asserted. As a simplification, it is assumed that the `TrafficIncident` objects are short lived. They represent an event that gets asserted and only those subscribers registered at that time are notified.

### 3.11.2 Using a Global Variable to Obtain Results

Using a global variable to accumulate results is a best practice, this approach yields simpler rule conditions than the following approaches.

Example 3–1 shows the `getSubscribers` function asserts a `TrafficIncident`, initializes a global variable with a `Map`, invokes the Rules Engine, and returns the result `Map`.

***Example 3–1 Obtaining Results with a RLFunction that Accesses a Global Variable***

```
Map alerts = null;

function getSubscribers(TrafficIncident ti) returns Map {
   try {
      alerts = new HashMap();
      assert(ti);
      run();
      return alerts;
   } finally {
      retract(ti);
```

```
      alerts = null;
   }
}
rule incidentAlert {
   if (fact TrafficIncident ti &&
      fact IncidentSubscription s &&
      s.highway == ti.highway &&
      s.direction == ti.direction) {
         alerts.put(s.subscriber, ti);
      }
   }
}
```

Example 3–2 shows Java code that invokes `getSubscribers` and prints out the results.

***Example 3–2 Sample Showing Results with Global Variable***

```
// An accident has happened
TrafficIncident ti = new TrafficIncident();
ti.setHighway("I5");
ti.setDirection("south");
ti.setIncident("accident");
ti.setWhen(new GregorianCalendar(2005, 1, 25, 5, 4));
ti.setDelay(45);

Map alerts = (Map)sess.callFunctionWithArgument("getSubscribers", ti);
Iterator iter = alerts.keySet().iterator();
while(iter.hasNext()) {
   String s = (String)iter.next();
   System.out.println("Alert " + s + " : " + alerts.get(s));
}
```

### 3.11.3 Using Container Objects to Obtain Results

In this approach, one or more objects are asserted into working memory to act as a container for results. The RL code that asserts the objects keeps the references handy. As rules fire, they can add results to the containers. A container object could be one of the Java Collection classes or it could be some application specific container object. When rule evaluation is complete the container objects can be inspected to access the results.

Example 3–3 uses a `java.util.Map` and shows that the subscriber (`key`) and the incident (`value`) are added to the map. The `getSubscribers` function asserts the `Map` and a `TrafficIncident`, invokes the Rules Engine, and returns the result `Map`.

---

**Note:** The `Map` is not reasserted even though it has been updated. In general, when an object that is being reasoned on is updated, it should be re-asserted. This use case represents an exception to that rule. The map, `alerts`, is just a container for results and its contents are not involved in the reasoning. Thus, it is okay not to re-assert it.

---

***Example 3–3 Obtaining Results with a Container Object***

```
function getSubscribers(TrafficIncident ti) returns Map {
   Map alerts = new HashMap();
   try {
      assert(alerts);
```

```
      assert(ti);
      run();
      return alerts;
   } finally {
      retract(alerts);
      retract(ti);
   }
}
rule incidentAlert {
   if (fact TrafficIncident ti &&
      fact IncidentSubscription s &&
      s.highway == ti.highway &&
      s.direction == ti.direction &&
      fact Map alerts) {
         alerts.put(s.subscriber, ti);
      }
   }
}
```

Example 3–2 shows Java code that invokes `getSubscribers` and prints out the results.

### 3.11.4 Using Reasoned On Objects to Obtain Results

In this approach, one or more objects are asserted into the rules engine working memory and the object references are retained in RL (in the `getSubscribers` function). Rule evaluation updates one or more of these objects. These objects are inspected after rule evaluation to determine the results.

In Example 3–4 the `TrafficIncident` class is modified to keep a `java.util.Set` of subscribers that need to be notified. Since the `TrafficIncident` object is being reasoned on, it is re-asserted. To avoid an infinite loop of rule firings for the same subscription, you need to use the subscribed method to test for matched incidents; this method prevents looping. The `subscribed` method returns `true` if a subscriber has already been added to the matched `TrafficIncident`. This is a common idiom in rules programming; a rule action updates a fact that is being reasoned on and, to avoid unwanted additional firings, you add a test to the condition that checks for the absence of that update.

Example 3–4 shows `getSubscribers` function asserts a `TrafficIncident`, invokes the Rules Engine, and creates and returns the result `Map`.

***Example 3–4 Obtaining Results with a Reasoned On Object***

```
function getSubscribers(TrafficIncident ti) returns Map {
   try {
      assert(ti);
      run();
      Map alerts = new HashMap();
      for (Iterator iter = ti.subscribers(); iter.hasNext(); ) {
         alerts.put(iter.next(), ti);
      }
      return alerts;
   } finally {
      retract(ti);
   }
}
rule incidentAlert {
   if (fact TrafficIncident ti &&
      fact IncidentSubscription s &&
```

```
            s.highway == ti.highway &&
            s.direction == ti.direction &&
            !ti.subscribed(s.subscriber)) {
               ti.addSubscriber(s.subscriber);
               assert(ti);
            }
      }
   }
```

Example 3–2 shows Java code that invokes `getSubscribers` and prints out the results.

# 4 Using XML Facts with Rule Author

This chapter provides a tutorial for working with Oracle Business Rules using XML documents (facts that are supplied in an XML document). This chapter also shows you how to create a rule enabled Java application that uses XML.

In the XML car rental How-To, driver data, supplied in an XML document specifies driver information, and the business rules determine if a rental company service representative should decline to rent a vehicle due to driver age restrictions (according to rental company business rules).

This chapter covers the following topics:


- Overview of Using XML Documents and Schemas with Rule Author
- Creating and Saving a Dictionary for the XML Car Rental Sample
- Defining a Data Model for the XML Car Rental Sample
- Defining Business Vocabulary for the XML Car Rental Sample
- Defining a Rule for the XML Car Rental Sample
- Customizing Rules for the XML Car Rental Sample
- Creating a Java Application with a Rule Session Using XML Facts
- Running the XML Car Rental Sample Using the Test Program


## 4.1 Overview of Using XML Documents and Schemas with Rule Author

Rule Author lets you import XML elements into a data model and lets you write rules using the XML elements in conditional expressions. For example, if you have an XML document that contains data associated with your application, and you have the schema associated with the XML document, then you can use Rule Author to define rules based on elements that you specify from the XML schema.

Starting with an XML schema, using XML documents with Rule Author involves the following steps:

1. XML schema processing: Rule Author generates Java classes from XML schema by running the supplied Java Architecture for XML Binding (JAXB) compiler to generate JAXB packages, classes, and interfaces for the XML Schema.
2. Using Rule Author you import XML elements into the data model in a dictionary.
3. Using Rule Author you define rules that reason on XML elements from an XML document. The process of writing rules for XML documents is very similar to writing rules using Java objects.

After you finish using Rule Author to create the rules, that use XML facts, you can write an application that reasons on XML documents. To accomplish XML document processing, you assert elements of an XML document into a Rules Engine session.

---

**Note 1:** JAXB sometimes maps XML construct names to different Java identifier names. For example, using JAXB, the XML name `my-element-name` becomes `myElementName`. Rule Author presents XML construct names so that you do not have to understand the JAXB generated XML-to-Java name mapping.

---

**Note 2:** If you want to use a JAXB binding compiler that is different from the Rule Author supplied implementation, you can manually perform the XML schema processing using your JAXB binding compiler and then use Java facts to import the generated Java packages and classes.

---

**See Also:** `http://java.sun.com/webservices/jaxb` for more information on JAXB.

## 4.2 Creating and Saving a Dictionary for the XML Car Rental Sample

To work with Rule Author you need to start with a dictionary. Rule Author stores rules and their associated definitions in a dictionary. To create or save a dictionary, you need to connect to a repository that stores the dictionary. As shipped, Rule Author supports two types of repositories: WebDAV (Web Distributed Authoring and Versioning) and file repository. In this section you create and save a dictionary for the XML car rental How-To.

The example in this chapter saves the dictionary to a WebDAV repository.

---

**Note:** To create the dictionary shown in this chapter, you can either create a new dictionary, using either a WebDAV repository or a file repository, or you can load the completed dictionary from the *$HowToDir*/`dict` directory supplied with the How-To.

Where $*HowToDir* is the directory where you install the How-To containing the XML car rental sample.

---

### 4.2.1 Connecting to a Rule Author Repository

Using Oracle Business Rules, a dictionary stores rules and the data model associated with the rules. You create and save dictionaries in a repository.

---

**Note:** Regardless of whether you choose to use a WebDAV or file repository, the repository must exist before you can connect to it. Rule Author does not create the repository for you.

See Appendix B, "Using Rule Author and Rules SDK with Repositories" for more information.

---

To connect to a repository, do the following:

1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. Select the `WebDAV` repository type in the **Repository Type** field.
4. Enter the URL to the WebDAV repository (see Figure 4–1). The URL must be in the form:

   `http://www.`*`fully_qualified_host_name`*`.com:7777/`*`repository_name`*

   > **Note:** In order for authentication to work, you must use a fully qualified host name in the URL.

***Figure 4–1 Rule Author Repository Connect Page***

   See Section B.1, "Working with a WebDAV Repository" for information on how to setup a WebDAV repository.

5. If you have a proxy server between the server on which Rule Author is running and the WebDAV server, specify the name and port number of the proxy server.
6. Click **Connect**.

   If you connect successfully, you receive a confirmation message.

### 4.2.2 Creating a Rule Author Dictionary

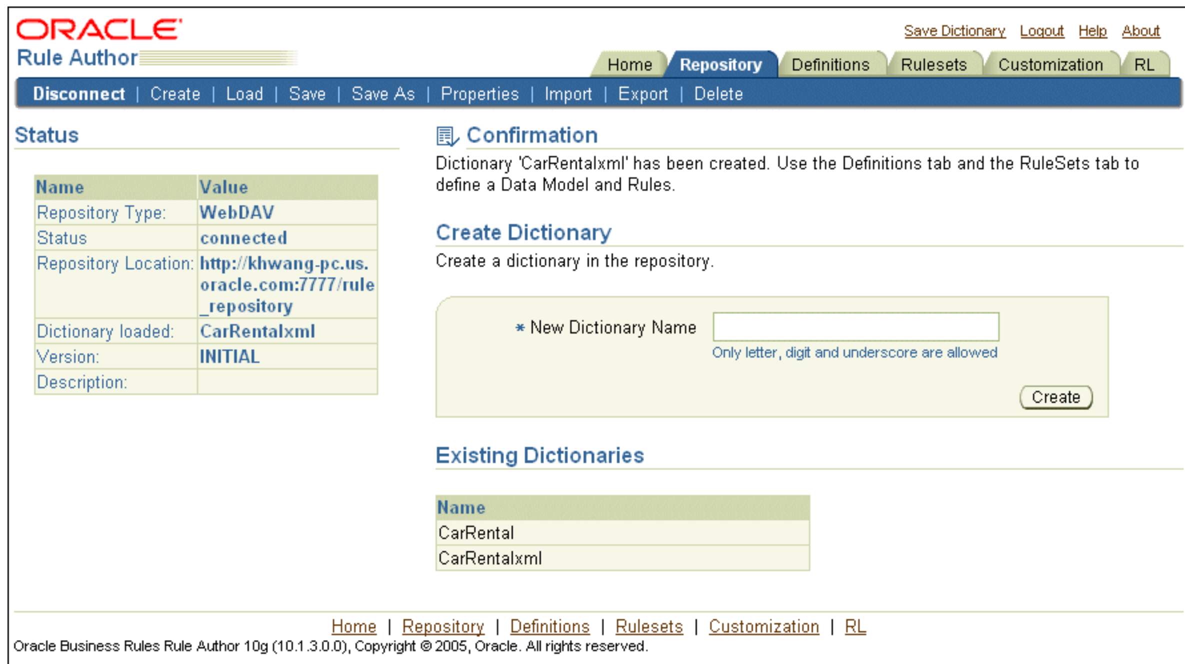
A Rule Author dictionary is the top-level container and the starting point for working with Rule Author. A dictionary usually corresponds to the rules portion of an application.

To create a dictionary, do the following:

1. Connect to a repository from the **Repository** tab.
2. Click the **Create** secondary tab.

3. Enter the dictionary name in the **New Dictionary Name** field. For this example enter `CarRentalxml`.
4. Click **Create**. After you click **Create**, Rule Author shows a status message (see Figure 4–2).

***Figure 4–2 Rule Author Create Dictionary (XML)***




## 4.2.3 Saving a Rule Author Dictionary with a Version

If you want to save to a different dictionary name or specify a version for the current dictionary, use the Save As area as follows:

1. Click the Repository tab.
2. Click the **Save As** secondary tab.
3. Enter a dictionary name in the **Dictionary** field, for example `CarRentalxml`.
4. If you want to specify a version that is associated with the dictionary, enter text in the **Version** field. For example `HowToxml`.
5. Click **Save As**. After clicking **Save As**, you should see a confirmation message in the status area (see Figure 4–3).

*Figure 4–3 Rule Author Save Dictionary (XML)*

ORACLE Rule Author

Save Dictionary | Logout | Help | About

Home | Repository | Definitions | Rulesets | Customization | RL

Disconnect | Create | Load | Save | Save As | Properties | Import | Export | Delete

**Status**

| Name | Value |
|---|---|
| Repository Type: | WebDAV |
| Status | connected |
| Repository Location: | http://khwang-pc.us.oracle.com:7777/rule_repository |
| Dictionary loaded: | CarRentalxml |
| Version: | HowToxml |
| Description: | |

**Confirmation**

Dictionary 'CarRentalxml (HowToxml)' has been saved.

**Save As**

Save contents to different dictionary or version

* Dictionary: CarRentalxml
Only letter, digit and underscore are allowed
* Version: HowToxml
Description:

Save As

Home | Repository | Definitions | Rulesets | Customization | RL

Oracle Business Rules Rule Author 10g (10.1.3.0.0), Copyright © 2005, Oracle. All rights reserved.

---

**Note:** Rule Author does not allow you to use **Save As** to overwrite a dictionary with the same name and version. If you want to overwrite a dictionary with the same name and version, do one of the following:

- Click **Save**.
- Delete the existing dictionary, then click the **Save As**.

---

## 4.2.4 Saving a Rule Author Dictionary

To prevent data loss, you should periodically save the dictionary. To save a dictionary, do one of the following:

- Click the **Repository** tab, then click the **Save** secondary tab.
- Click the **Save Dictionary** link at the top of the page.

After performing either of the preceding actions, click **Save** on the Save Dictionary page. After clicking **Save**, you should see a confirmation message in the status area. For example:

```
Dictionary 'CarRental(HowTo)' has been saved
```

---

**Note:** You should save the dictionary periodically as you work since Rule Author sessions time out after a period of inactivity.

---

**See Also:** "Rule Author Session Timeout" on page A-2 for details on configuring the Rule Author session timeout.

## 4.3 Defining a Data Model for the XML Car Rental Sample

Before working with rules you need to define a data model. A data model contains business data definitions for facts or data objects used in rules, including: Java class fact types, XML fact types, and RL Language fact types. In this section you work with a data model that includes XML fact types.

This section covers the following topics:

- Using XML Schema as Facts in the XML Car Rental Sample
- Adding XML Facts for the Car Rental Sample (XML Schema Processing)
- Importing XML Schema Elements to a Data Model
- Viewing XML Facts in a Data Model
- Saving the Current State of XML Fact Definitions

**See Also:** "Adding Java Classes and Packages to Rule Author" on page 2-9

### 4.3.1 Using XML Schema as Facts in the XML Car Rental Sample

The XML sample includes the `carrental.xsd` file in the *$HowToDir*/`data` directory. This file specifies the schema for the XML car rental sample that uses XML documents to assert facts.

Where $*HowToDir* is the directory where you installed the XML car rental How-To.

### 4.3.2 Adding XML Facts for the Car Rental Sample (XML Schema Processing)

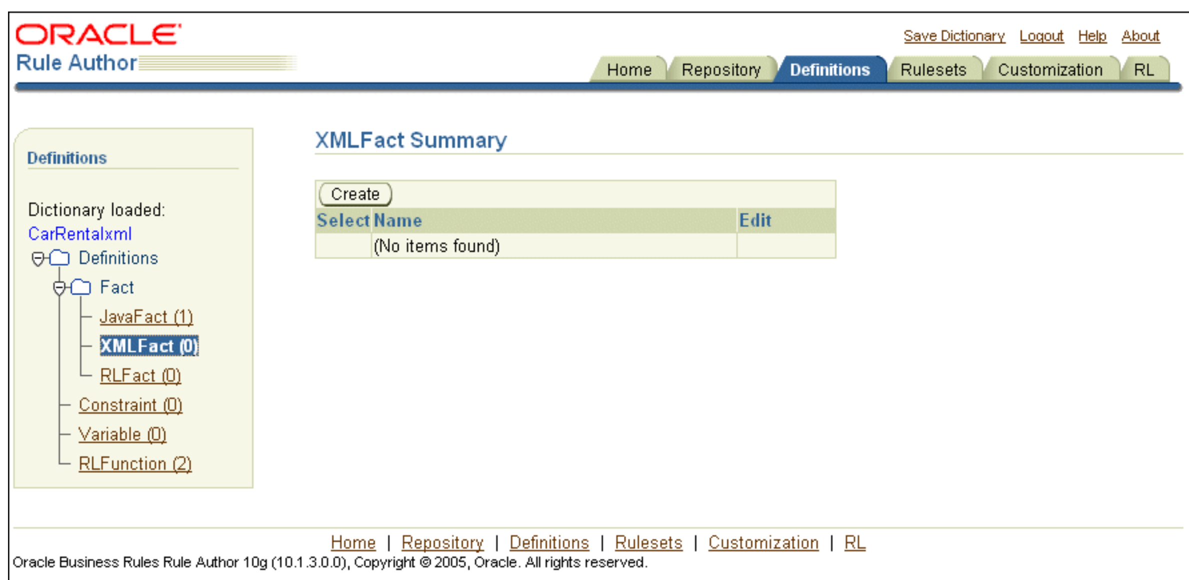
Before you can use XML elements in a data model, Rule Author needs to generate the JAXB classes representing the XML elements. This step generates the JAXB classes and makes the generated classes and packages associated with the XML schema visible to Rule Author.

To use Rule Author to prepare the sample XML car rental schema, do the following:

1. Go to Step 2 if you just created the CarRentalxml dictionary. Click the **Repository** tab and load the CarRentalxml dictionary.
2. Click the **Definitions** tab. The navigation tree shows the Definitions folder that contains the available definitions.
3. The Definitions folder in the tree contains the Facts folder that includes the available fact types: **JavaFact**, **XMLFact**, and **RLFact**.

   Click **XMLFact** to view the XMLFact Summary page (see Figure 4–4).

***Figure 4–4 Rule Author Definitions XML Fact Summary Page***




4. Click **Create**. This shows the XML Schema Selector page.
5. On the XML Schema Selector page, in the **XML Schema** field enter either the path or HTTP URL to the schema. For example:
   - *$HowToDir*`/data/carrental.xsd`, where *$HowToDir* is the directory where you installed the XML How-To.
   - `http://www.myCompany.com/xsd/product.xsd`

   If you choose to access the schema with a URL, you must set the following system properties:

   ```
   proxyHost = $YourProxyHost
   proxyPort = $YourProxyPort
   proxySet = true
   ```

   For example:

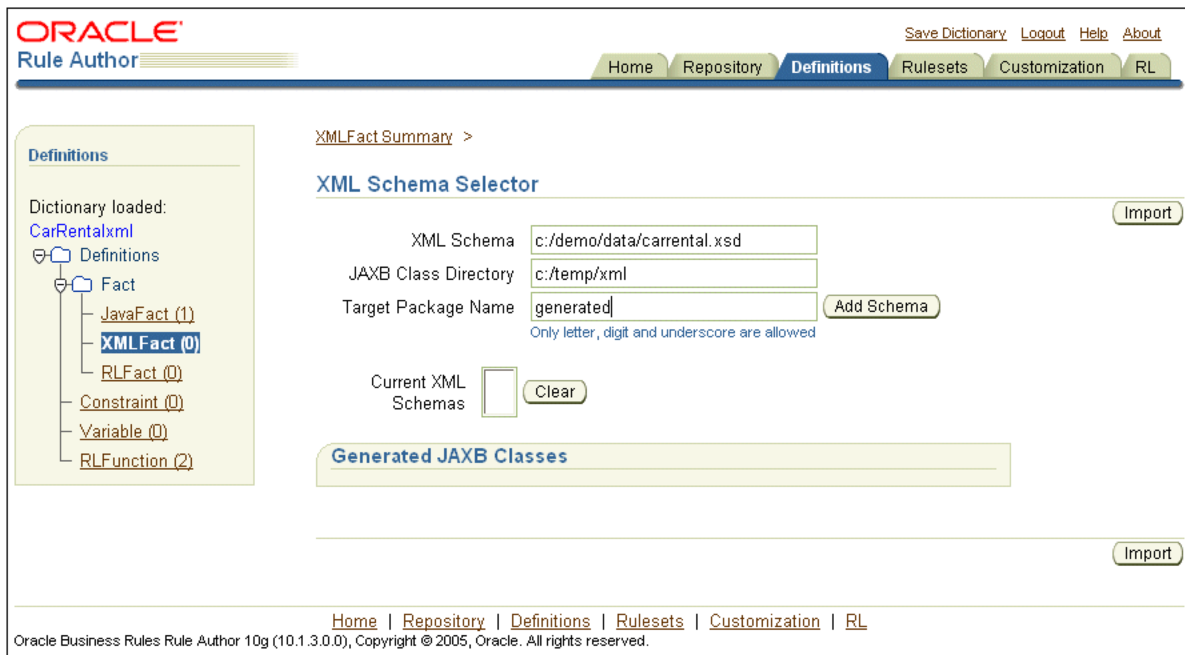
   ```
   -DproxyHost=www-proxy.myCompany.com -DproxyPort=80 -DproxySet=true
   ```

   For more information on setting system properties in an OC4J instance, see *Oracle Containers for J2EE Configuration and Administration Guide*.
6. In the **JAXB Class Directory** field enter the directory where you want Rule Author to store the JAXB generated classes.

   Note: the directory that you specify must be writable.
7. Enter a value for the **Target Package Name** field. If you leave this field empty, the JAXB classes package name is generated from the XML schema's target namespace using the default JAXB XML-to-Java mapping rule. For example, the namespace rules.oracle.com is mapped to com.oracle.rules.

   The value you enter specifies the generated classes package name. For example, `generated` (for this example, we use the name generated, there is nothing special about the name generated. This value specifies the name of the package, directory, where the generated classes are placed). See Figure 4–5.

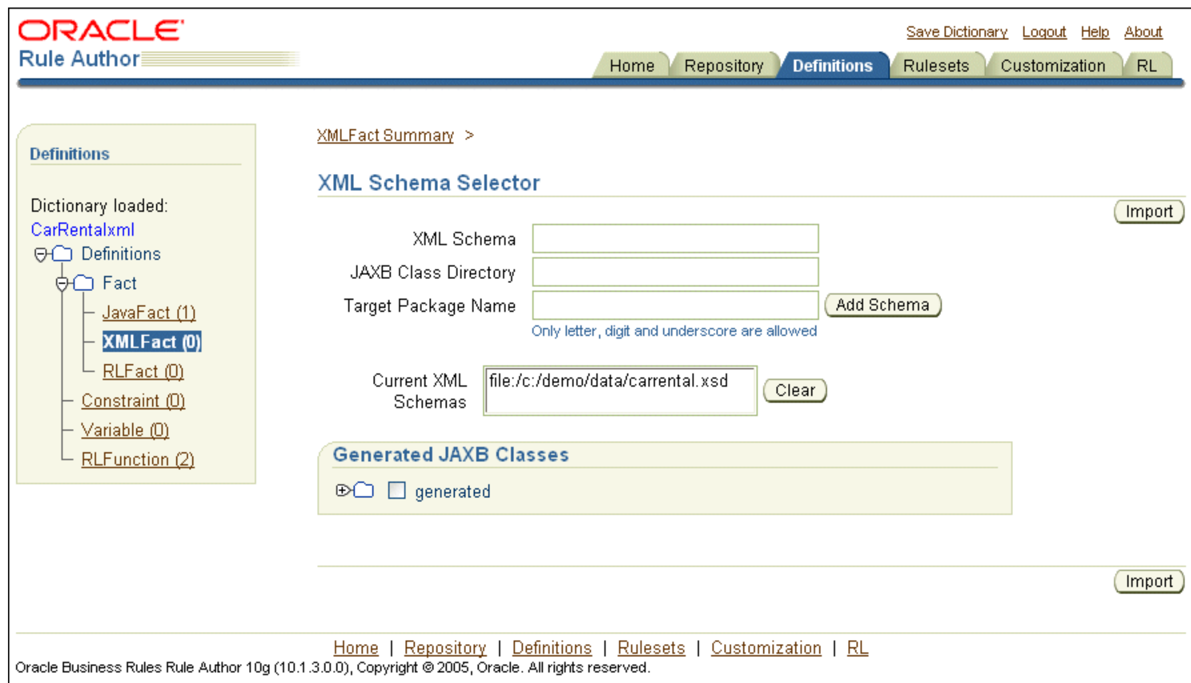
***Figure 4–5 Rule Author XML Schema Selection Page***




8. Click **Add Schema**. This step requires some processing to compile the JAXB, so depending on the size of the Schema, you may need to wait for a period of time for this step to complete.

   When this completes the page shows cleared Add Schema text entry fields, and Rule Author updates the **Current XML Schemas** field, and shows the Generated JAXB Classes area (see Figure 4–6).

***Figure 4–6 Rule Author Definitions XML Schema Selector After Adding XML Schema***




**See Also:** "Creating and Saving a Dictionary for the XML Car Rental Sample" on page 4-2

## 4.3.3 Importing XML Schema Elements to a Data Model

This step brings the JAXB generated classes representing the XML schema elements into the data model (from the sample schema `carrental.xsd`). Select the XML elements to import into the data model using the Class Selector page from the **Definitions** tab.

Oracle Rules Engine binds an XML schema to Java classes by using JAXB. In most cases, the default bindings generated by the Oracle JAXB binding compiler are sufficient to meet your needs. There are cases, however, when you may want to modify the default bindings. For example:

1. Name collision
2. Invalid Java identifiers mapped from non-English tag names

Please refer to Oracle JAXB documentation for more details about customizing the default XML-to-Java mapping.

To add `DriverType` from the schema to the data model, do the following:

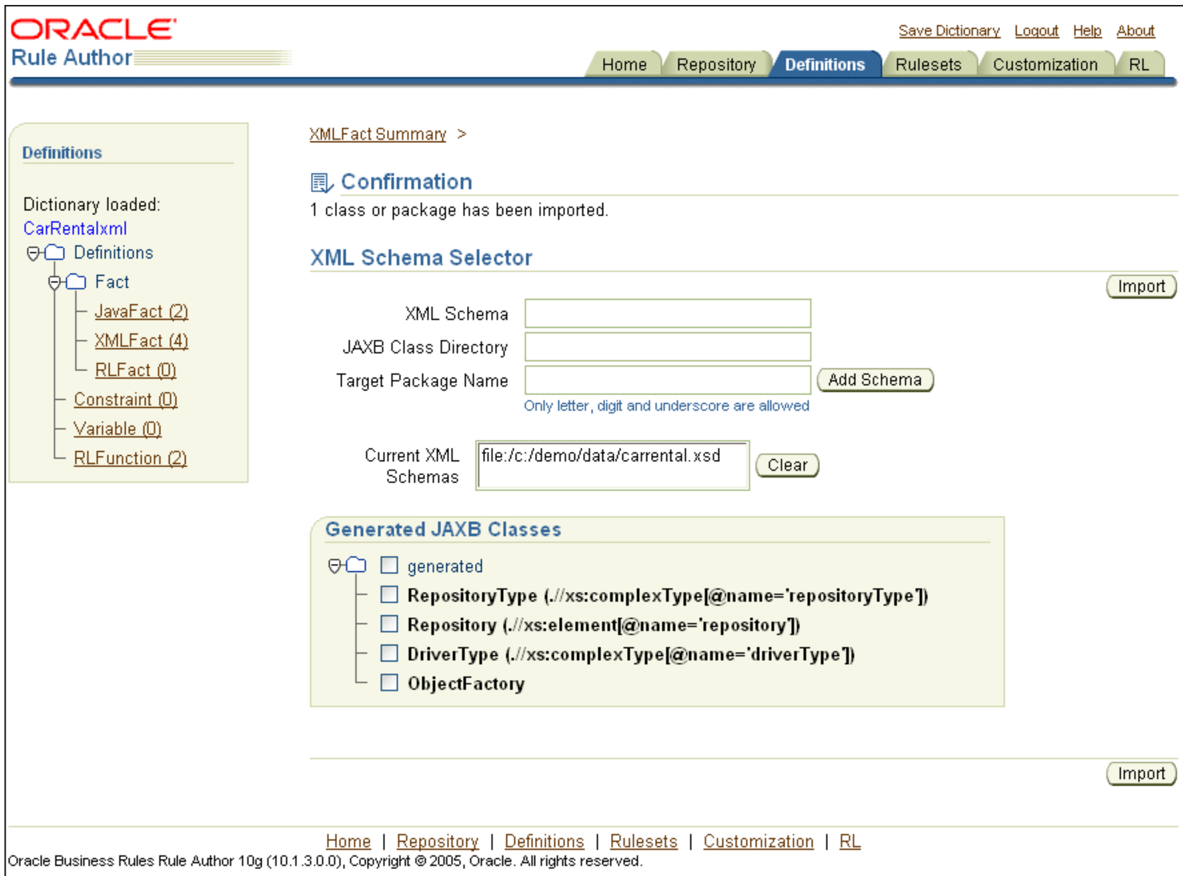
1. Click the **Definitions** tab to view the Definitions page.
2. Click the **XMLFact** folder in the navigation tree.
3. Click **Create** on the XMLFact Summary page. This shows the XML Schema Selector page.
4. In the Generated JAXB Classes box on the XML Schema Selector page expand the navigation tree until you see **DriverType**.
5. Select the generated folder check box.

6. Click **Import**. Rule Author shows a confirmation message: "1 class or package has been imported" (see Figure 4–7).

7. Expand the Generated node in the Generated JAXB Classes area to see the imported classes (see Figure 4.7).

> **Note:** After an element is imported, the element is shown in bold.

***Figure 4–7 Rule Author XML Schema Selector with Confirmation Message***




Notes for Adding XML Schema to Rule Author:

- The Classes navigation tree is rendered on demand (to improve performance). Thus, a child node is rendered only if its parent node is expanded. It is a good practice to keep only the nodes of interest expanded.
- On Windows systems, you can use a "\" or a "/" as a path separator. Rule Author accepts either path separator.
- A corresponding XML construct name is displayed next to each Java class so that you know where the Java class is generated from (using the XML Schema names). If you want to import the whole package into the data model, check the package name and click **Import**.
- Using Rule Author, importing an XMLFact means the same thing as a Java import statement. That is, the JAXB classes and their methods become visible to Rule

Author. Rule Author does not copy the classes into the data model or into the dictionary.

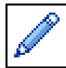
- Do not use RL reserved words in Java package names. For more information, see Section D.8, "Using RL Reserved Words as Part of a Java Package Name"

### 4.3.4 Viewing XML Facts in a Data Model

To view the XML Facts in a data model, including any JAXB generated classes or packages that you import, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. Expand the Facts folder and click the **XMLFact** node in the navigation tree to display the XMLFact Summary page.

   For the XML car rental sample this shows the XMLFact table that includes the imported classes DriverType, RepositoryType, Repository, and ObjectFactory.
3. Click the edit icon to view the XML Fact Properties and Methods.



---

**Note:** Importing a Java class does not cause all of its super classes and classes associated through fields and methods to be imported into the data model. In order to access these correctly, they must be explicitly imported into the data model.

---

**See Also:** See "Specifying Visibility and Object Chaining for Rule Author Drop Down Lists" on page 3-9 for details on the **Visible** and **Expand** fields in the XML Fact Properties and Methods table.

### 4.3.5 Saving the Current State of XML Fact Definitions

While working on a data model from the Definitions tab and when you complete your work, it is important to save the dictionary. You can save the dictionary in two ways:

1. Click the **Repository** tab, then click the **Save** secondary tab.
2. Click the **Save Dictionary** link at the top of the page.

After performing either of the preceding actions, click **Save** on the Save Dictionary page.

**See Also:** "Saving a Rule Author Dictionary" on page 2-8

## 4.4 Defining Business Vocabulary for the XML Car Rental Sample

The business vocabulary allows business analysts to create rules using familiar names rather than using an XML name or a Java package name, class name, method name, or member variable name. You use the Rule Author aliases feature to specify the business vocabulary. In this step you only need to define the business vocabulary for the business objects that you expect to use in rules. In addition, you can use the Rule Author **Visible** box to specify the properties and methods that show up in Rule Author lists when you create rules from the **RuleSets** tab.

This section covers the following topics:

- Specifying the Business Vocabulary for XML Fact Definitions
- Specifying the Business Vocabulary for Functions

### 4.4.1 Specifying the Business Vocabulary for XML Fact Definitions

To specify the business vocabulary for XMLFact definitions, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. Expand the Facts folder and click the **XMLFact** node in the navigation tree to display the XML Fact Summary page. For the XML car rental sample this shows a table that includes the class `generated.DriverType` (if you specify a package name other than generated, then the package name is different than `generated`).
3. Click the edit icon for `driverType`. This shows the XML fact page.
4. At the top of the XML fact page, in the **Alias** field enter `DriverData`.
5. For the `age` entry in the Properties table, specify the desired alias. For example, enter `DriverAge` in the **Alias** field.
6. For the name entry in the Properties table, specify the desired alias. For example, enter `DriverName` in the **Alias** field.
7. Click **OK** or **Apply**.
8. Save the dictionary.

> **Note:** Be sure to click **OK** or **Apply** after making changes. Otherwise, Rule Author does not save your changes.

Notes for specifying the business vocabulary for XML fact definitions:

- On the XML fact page, you can specify that Rule Author shows methods or properties one level above a specified method or property, in superclass chain, by selecting the **Expand** box for the method or property on the XML Fact page. The Expand box is shown in the **Expand** field of the Properties and Methods area. The checkbox is only shown for methods or properties that include a superclass (Rule Author does not show the **Expand** box for primitive types).
- On the XML fact page you can specify that properties or classes are not visible in Rule Author list boxes. Deselect the **Visible** checkbox to specify that an object is not visible in Rule Author list boxes (by default objects are visible).
- The XML Fact page includes the **XML Name** and **Generated From** fields that show the Java class was generated from an XML schema.

  For example, `//complexType[@name='driverType']` **XML Name** shows that the class is generated from an XML complex type named `driverType`. The **Generated From** field shows the name of the XML schema that generated the JAXB classes for the XML Fact.
- Make sure the **Support XPath Assertion** box is checked for all XML FactTypes. For more information, see Section D.10, "XML Facts not Asserted at Runtime".

### 4.4.2 Specifying the Business Vocabulary for Functions

To specify the business vocabulary for an RL Language function, do the following:

1. Click the **Definitions** tab to view the Definitions page.

2. Click RLFunction in the Definitions folder in the navigation tree to display the RLFunction Summary page. For the XML car rental sample, this shows a table that includes the functions, `DM.assertXPath` and `DM.println`.
3. For the `DM.println` function, click the edit icon in the **Edit** field to view details.
4. In the **Alias** field, under the **Name** field, enter an alias. For example, enter `PrintOutput` in the **Alias** field.

> **Note:** There is also an Alias field in the Function Arguments table. For this example we are not changing the function arguments alias.

5. Click **OK** or **Apply**.
6. Save the dictionary.

## 4.5 Defining a Rule for the XML Car Rental Sample

In this section you define a rule for the XML car rental sample.

This section covers the following topics:

- Creating a Rule Set for the XML Car Rental Sample
- Creating a Rule for the XML Car Rental Sample

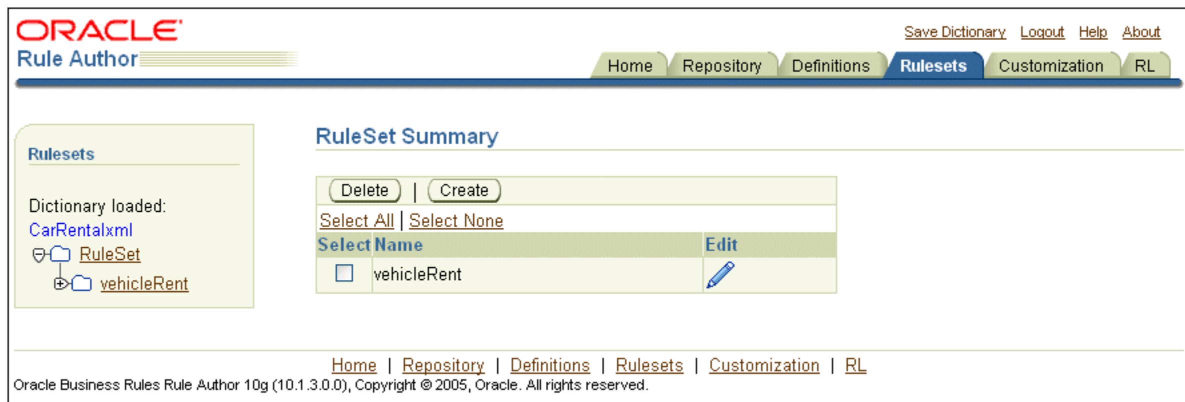
### 4.5.1 Creating a Rule Set for the XML Car Rental Sample

Before you can create a rule you need to create a rule set. A rule set is a container for rules.

To create a rule set, do the following:

1. Click the **Rulesets** tab.
2. Click the **RuleSet** node in the navigation tree.
3. On the Ruleset Summary page, click **Create**. This displays the Ruleset page.
4. Enter a name in the **Name** field. For example, enter `vehicleRent`.
5. Optionally enter text in the **Description** field. For example, enter `vehicle rent rule set`.
6. Click **OK**. This creates the `vehicleRent` rule set. After you create the rule set, the tree shows the new entry, as shown in Figure 4–8.
7. Save the dictionary.

*Figure 4–8 Rule Author RuleSet Summary Page*




## 4.5.2 Creating a Rule for the XML Car Rental Sample

After creating a rule set, you can create rules within the rule set. In this section, you create the UnderAge rule. The UnderAge rule tests the following:

If the driver's age is younger than 21, then decline to rent

The UnderAge rule contains a single pattern for the Rules Engine to match, and the rule includes a test that is applied to the pattern.

The following actions are associated with the UnderAge rule:

- Print "Rental declined", the name of the driver matched and the message, "Under Age, age is: " and the driver’s age.
- Retract the matched driver fact from the rule session.

### 4.5.2.1 Adding the Under Age Rule for the XML Car Rental Sample

To use Rule Author to add the UnderAge rule, do the following:

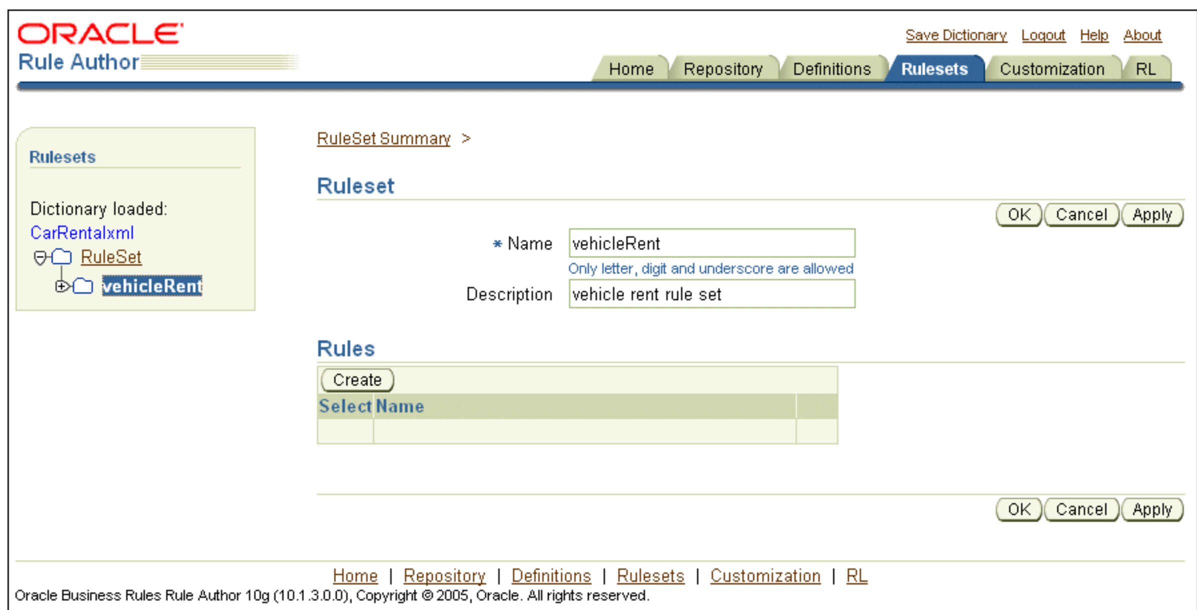
1. Click the **RuleSets** tab. The navigation pane displays the RuleSet folder that contains the vehicleRent rule set that you created in Section 4.5.1.
2. Click the **vehicleRent** node in the navigation tree. This displays the Ruleset page, with a table listing rules (see Figure 4–9).

---

**Note:** If there are no rules, the Rules table is empty.

---

*Figure 4–9 Rule Author RuleSet Page Showing the Create Button*

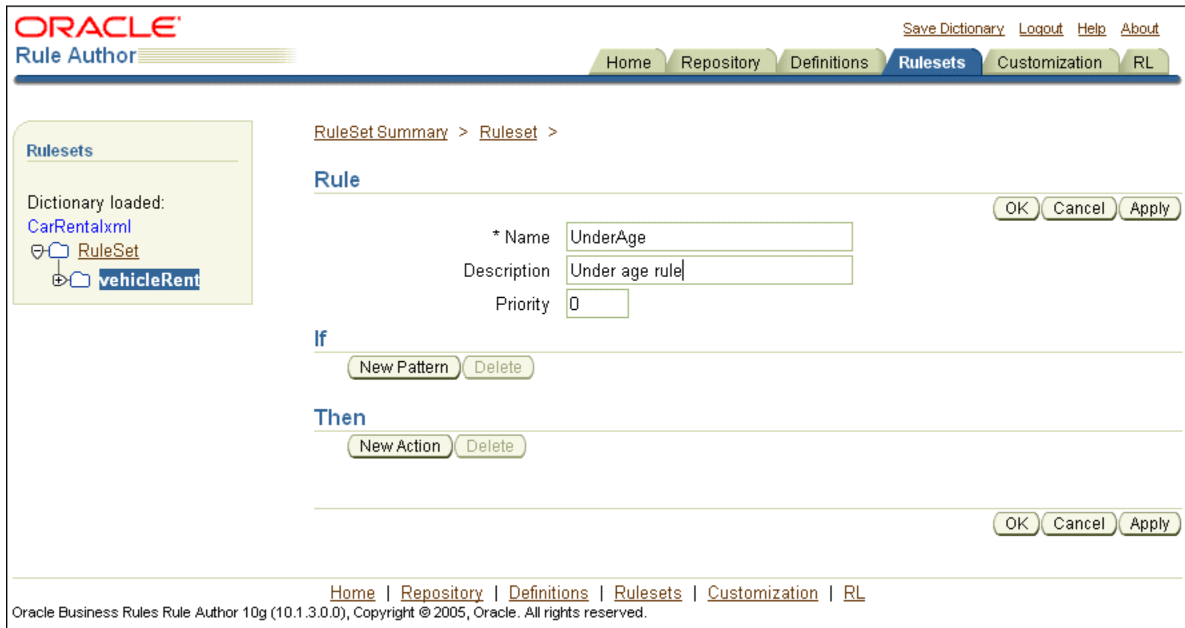



3. Click **Create**. This displays the Rule page.
4. On the Rule page enter `UnderAge` in the **Name** field.
5. On the Rule page enter 0 in the **Priority** field.

> **Note:** The **Priority** field determines which rule to act upon, and in what order, if more than one rule applies. Often in applications that use rules, the rules are be applied in any order until a decision is reached, and setting a priority is not required.

6. Enter a description in the **Description** field (see Figure 4–10).

*Figure 4–10 Rule Author Rule Page*




### 4.5.2.2 Adding a Pattern to the Under Age Rule (XML)

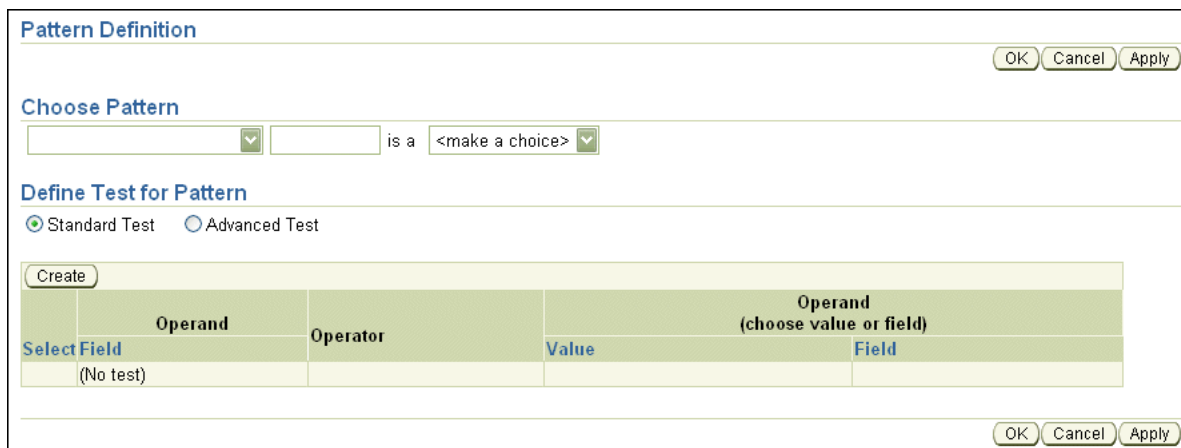
When the Rules Engine runs, it uses the rules to check the available facts for matching patterns. To add a pattern for the UnderAge rule, do the following:

1. Click **New Pattern** in the **If** box on the Rule page. This displays the Pattern Definition page.

> **Note:** If the Pattern Definition page does not appear, you may have popup blocking enabled on your browser. Popup blocking must be disabled in order to use Rule Author.

2. The Pattern Definition page contains two areas: Choose Pattern and Define Test(s) for Pattern (see Figure 4–11).

*Figure 4–11 Rule Author Pattern Definition Page*



3. Under **Choose Pattern**, in the first box select the first entry, which is blank.

   This box specifies that the rule should fire each time there is a match (for all matching drivers). One alternative value, `There is at least one case`, selects one firing of the rule if there is at least one match (one such driver). The alternate value, `There is no case`, specifies the rule fires once if there are no such matches (no matching drivers).

4. The next text area under **Choose Pattern** lets you enter a temporary name for the matched fact.

   Enter `driver` in this field (this defines the "pattern bind variable name").

   This field lets you test multiple instances of the same type in a single rule. For example, this lets you compare a driver with other drivers, using the specified name, in a comparison such as `driver1.age` > `driver2.age`.

5. The third box contains the text, `<make a choice>`, this shows the available fact types. In this box select `DriverData`.

6. Click **OK** to save the pattern definition. This closes the Pattern Definition page.

7. Click **OK** to save the rule.

> **Note:** Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you navigate to a different rule set or select a different tab before you click **OK** or **Apply**, Rule Author discards your pattern definition changes.

8. Save the dictionary.

Without any tests defined on the pattern, the action that you define would apply to all drivers. To define tests for patterns, continue, as shown in, Section 4.5.2.3.

**See Also:** "Adding Actions for the Under Age Rule (XML)" on page 4-19

#### 4.5.2.3 Defining Tests for Patterns with the Under Age Rule (XML)

To add a test for a pattern, do the following:

1. From the **Rulesets** tab, in the navigation tree click the rule where you want to add a test. For this example click the UnderAge rule.

2. In the If table on the rule page, select the pencil icon to bring up the Pattern Definition page for this rule.

3. On the Pattern Definition page, select the **Standard Test** button, then click **Create** (see Figure 4–12).

***Figure 4–12 Rule Author Rule Pattern Definition Page with New Test Fields***

Standard pattern testing is only valid for AND expressions. Additionally, no grouping is allowed, and functions with parameters are not allowed. However, the use of constraints is allowed for customization. Advanced pattern testing does not have the restrictions of standard pattern testing, but the use of constraints is not allowed. Advanced expressions are not directly RL Language because aliases are used instead of variable names.

For more information, see Section 3.7.1, "Using the Advanced Test Expression Option".

4. In the **Operand** column, from the **Field** box, select `driver.DriverAge`.
5. In the **Operator** column, select < (less than).
6. In the next **Operand** column, in the **Value** box enter 21. Do not enter a value in the **Field** box.
7. Next to the **Value** and **Field** boxes is a drop-down list containing the fixed values `Any` and `Fixed` (see Figure 4–13).

   These values are called constraints, and they are used to enable or disable customization for this field. Use the value `Fixed` to make the field read-only, which specifies that no customization is allowed for this field. Select the value `Any` to specify that Rule Author should allow changes to the value. Setting a value of `Any` allows for rule customization (which supports modifications by non-technical users). You can also define constraints that allow you to limit the allowed values.

   Select `Any` as the constraint for the **Value** field.

***Figure 4–13 Rule Author Pattern Definition Page with Values for Under Age Rule***

Pattern Definition

OK Cancel Apply

Choose Pattern

driver is a DriverData

Define Test for Pattern

Standard Test Advanced Test

Delete | Create

Select All | Select None

| Select | Operand Field | Operator | Operand (choose value or field) Value | | Field | |
|---|---|---|---|---|---|---|
| ☐ | driver.DriverAge | < | 21 | Any | <make a choice> | Fixed |

OK Cancel Apply

8. Click **OK** to save your changes and close the Pattern Definition page.
9. On the Rule page, click **OK** or **Apply**.

> **Note:** Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you do not click **OK** or **Apply**, Rule Author does not save your work on the rule.

10. Save the dictionary.

> **See Also:** "Customizing Rules for the XML Car Rental Sample" on page 4-22

### 4.5.2.4 Adding Actions for the Under Age Rule (XML)

Actions are associated with pattern matches. When a rule's "If" portion matches, the Rules Engine executes the "Then" portion to run the rule's action.

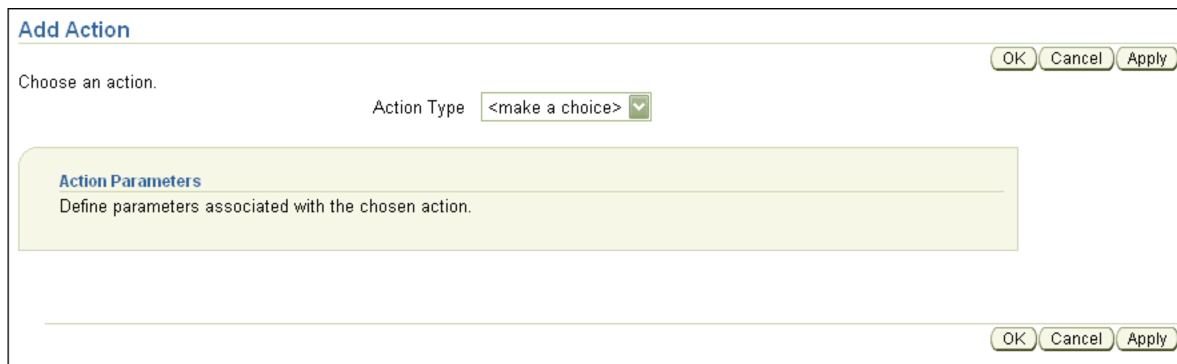
In this section, you add two actions for the UnderAge rule. The first action prints the result. The second action retracts the driver fact from the Rules Engine. You might want to retract a fact for a number of reasons, including:

- If you are done with the fact, and you want to remove it from the Rules Engine.
- The action associated with the rule changes the state, so that the fact needs to be retracted to represent the current state of the Rules Engine.

To add the action that prints the result for a match of the UnderAge rule, do the following:

1. Click the **Rulesets** tab.
2. In the tree, click the UnderAge node under the vehicleRent folder.
3. Click **New Action** on the Rule page in the **Then** box. This brings up the Add Action page (see Figure 4–14).

*Figure 4–14 Rule Author Add Action Page*




4. Select the `Call` item from the **Action Type** box. This shows the **Action Parameters** box.
5. Choose `PrintOutput` from the **Function** box (if you did not define an alias for `println`, then this is `DM.println`). This shows the function arguments box.
6. Enter the argument value in the **Argument Value** field (see figure Figure 4–15):

```
"Rental declined" + driver.DriverName + " Under age,age is:" + driver.DriverAge
```

---

**Note:** Rule Author uses a Java like syntax for expressions. The RL Language defines the complete expression syntax.

---

---

**Note:** You can also select the edit icon in the Wizard field to use the expression builder wizard to enter the expression. This provides you with more space to write expressions. This also provides an easier and more accurate way to enter variables, since the expression builder presents a list showing the available variables.

---

***Figure 4–15 Rule Author Add Action Page for Under Age Rule***

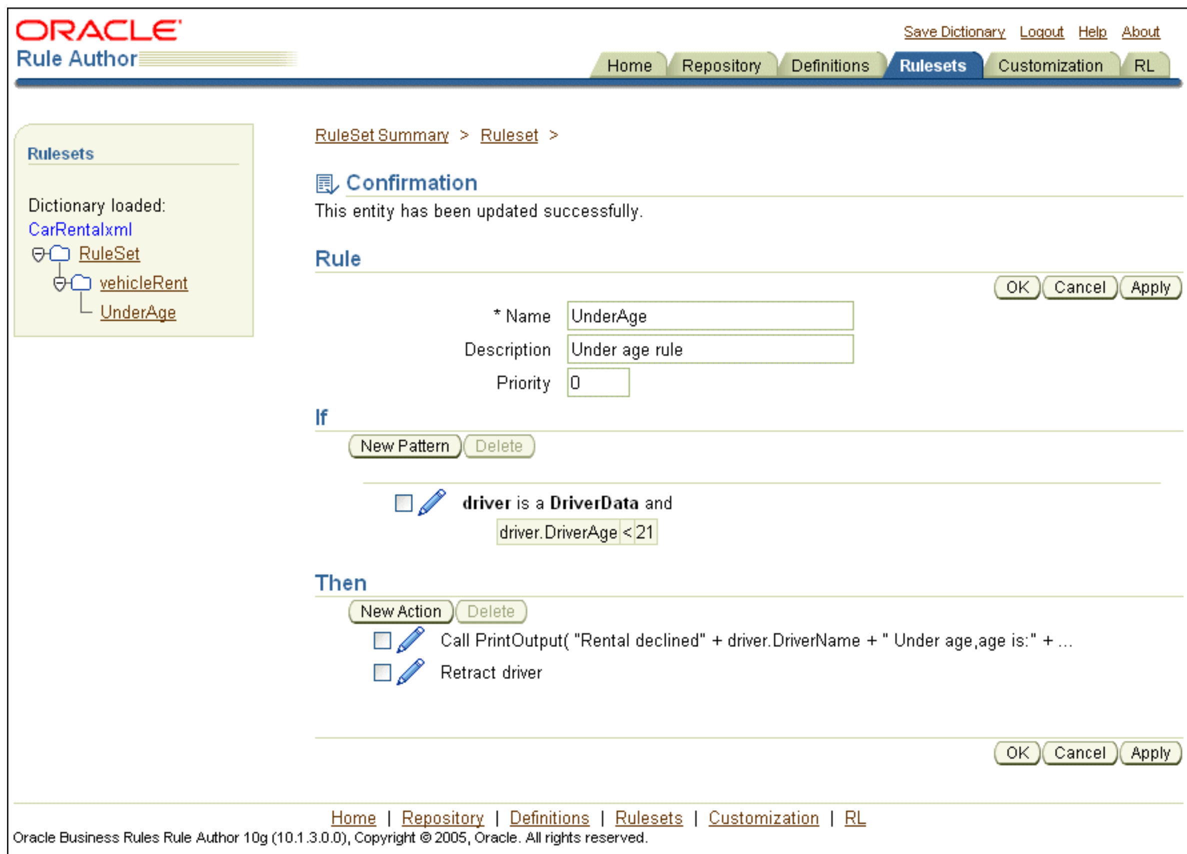


7. Click **OK** to save your changes and close the Add Action page.
8. Click **OK** or **Apply** on the Rule page.
9. Save the dictionary to save your work.

Next, add the retract action for the UnderAge rule. Perform the following steps to add this second action for the rule:

1. Click the **Rulesets** tab.
2. Click the UnderAge node under the vehicleRent folder.
3. On the Rule page, click **New Action** from the **Then** box. This brings up the Add Action page.
4. Select `Retract` from the **Action Type** box. This shows the **Action Parameters** box.
5. Select `driver` from the Fact Instance box. The pattern name `driver`, when used in the action refers to a single instance which was matched by the pattern.
6. Click **OK** to save your changes and close the Add Action page.
7. Click **Apply** on the Rule page to receive a confirmation message (see Figure 4–16).
8. Save the dictionary.

***Figure 4–16 Rule Author Under Age Rule with Pattern and Actions***




---

**Note:** When you add actions to rules, you can only add new actions sequentially. If an action depends on the results of a previous action, then the order in which you add the actions is significant.

---

**See Also:** The *Oracle Business Rules RL Language Reference Guide*

## 4.6 Customizing Rules for the XML Car Rental Sample

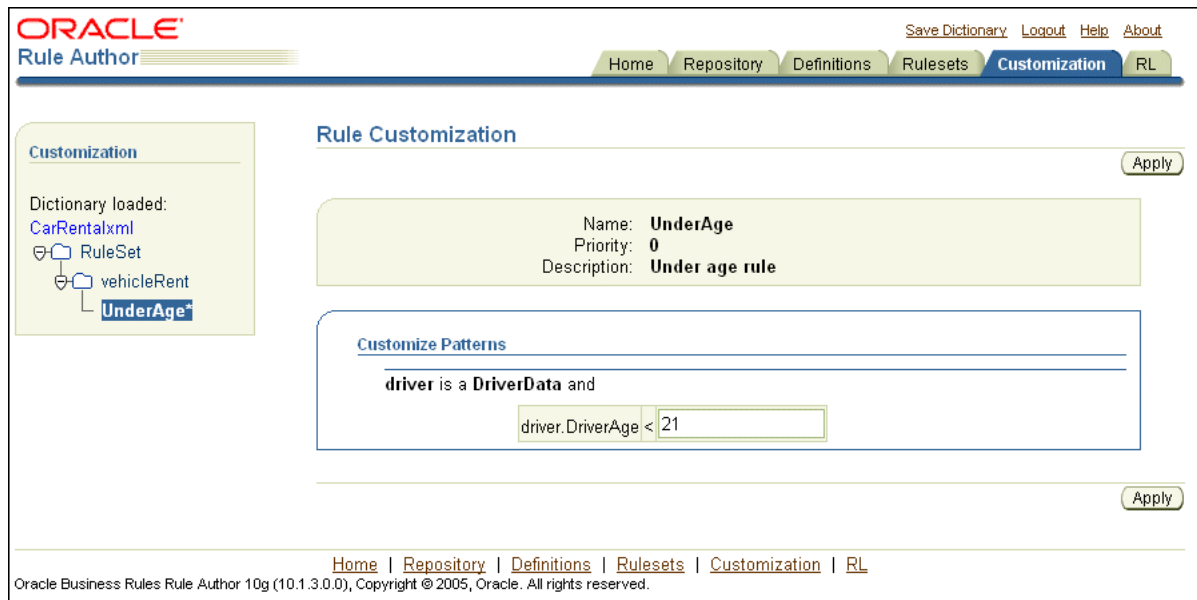
The Rule Author **Customization** tab is designed for business users. Rule developers use the **Allowed Values** field on the Pattern Definition page, which is available from the **Ruleset** tab, to specify if customization is allowed. When customization is allowed you can specify a range of values for the customizable value. Then, business users may change values using the **Customization** tab.

In this example, the UnderAge rule can be modified on the **Customization** tab to change the age of an under age driver (for this sample we do not limit values, and specify that any value is valid).

To change the UnderAge rule, use the **Customization** tab as follows:

1. Click the **Customization** tab. The navigation pane displays the vehicleRent folder with the UnderAge node followed by a "*", which indicates that the rule is customizable.
2. Click the node for the UnderAge rule (see Figure 4–17).

***Figure 4–17 Rule Author Rule Customization Page for Under Age Rule***




3. On the Rule Customization page the Customize Patterns box contains an editable text entry field for the test `driver.DriverAge < 21`.

   Enter 19 in this field (change the value from 21 to 19).

4. Click **Apply**.
5. Save the dictionary.

After you save the dictionary, you are done creating the data model and the rules for the XML car rental sample.

> **See Also:** Defining Tests for Patterns with the Under Age Rule (XML) on page 4-17

## 4.7 Creating a Java Application with a Rule Session Using XML Facts

After you create and save a Rule Author dictionary that contains a data model and a rule set with rules, you can rule enable an existing Java application or create a new rule enabled Java application. This section shows you the steps for creating a rule enabled application.

This section covers the following:

- Importing the Rules SDK and Rules RL Classes
- Creating a JAXB Context and Unmarshalling the XML Document
- Loading a Dictionary with Rules SDK
- Loading a Ruleset and Generating RL Language for Data Model and Rule Set
- Initializing and Executing a Rule Session
- Asserting XML Data from Within a Rule Session
- Using the Run Function with a Rule Session

For the complete code for this sample application, see `TestXML.java` in the *$HowToDir*`/src/carrental` directory. Where *$HowToDir* is the directory where you installed the XML How-To.

---

**Note 1:** If you have completed the Java car rental example from Chapter 2, the differences in this example are that you need to create a JAXB context, and when you add facts to a rule session you use the assertXPath function.

---

---

**Note 2:** The instructions in the preceding sections of this chapter enabled you to create and save a WebDAV repository and dictionary named `CarRentalxml`. The car rental example supplied in the How-To sample code uses a file repository with a dictionary also named `CarRentalxml`. The dictionary contents in the WebDAV repository you created in this chapter and the file repository in the How-To sample are identical.

The How-To sample code contains code for both WebDAV and file repositories, but only the file repository is described in detail. The How-To sample uses a file repository for portability, but this sample can be modified to use the WebDAV repository you created in the proceeding sections.

---

### 4.7.1 Importing the Rules SDK and Rules RL Classes

The first step when writing an Oracle Business Rules enabled program is to import certain required classes. Example 4–1 shows the imports from the `TestXML.java` application for the XML car rental sample.

***Example 4–1 Required Imports for XML Car Rental Sample with Rules SDK***

```
package carrental;

import java.io.File;
import java.util.List;
import java.util.ArrayList;
import java.util.Properties;
import javax.xml.bind.*;

import oracle.rules.sdk.ruleset.RuleSet;
import oracle.rules.sdk.repository.RuleRepository;
import oracle.rules.sdk.repository.RepositoryManager;
import oracle.rules.sdk.repository.RepositoryType;
import oracle.rules.sdk.repository.RepositoryContext;
import oracle.rules.sdk.dictionary.RuleDictionary;
import oracle.rules.sdk.exception.RepositoryException;

import oracle.rules.rl.RuleSession;
```

### 4.7.2 Creating a JAXB Context and Unmarshalling the XML Document

Using the JAXB generated classes either generated using Rule Author or manually, you first need to specify a JAXB context and unmarshall an XML document that conforms to the schema. Example 4–2 shows this code from `TestXML.java`.

***Example 4–2 Unmarshalling an XML Document***

```
JAXBContext jc = JAXBContext.newInstance("generated");
Unmarshaller um = jc.createUnmarshaller();
String fs = System.getProperty("file.separator");
String xmlpath = "data" + fs + "carrental.xml" ;
Object root = um.unmarshal(new File(xmlpath));
```

### 4.7.3 Loading a Dictionary with Rules SDK

When building a rule enabled Java application, do the following to access a dictionary and specify a rule set (see Example 4–3):

1. Use a Rules SDK `RuleRepository` object to access one or more dictionaries. The parameter to the `getDefaultRepository` method specifies the location of the dictionary directory.
2. Use a Rules SDK `SecurityInfo` object to specify the credentials for accessing a dictionary (the default rule storage plug-in uses file-based storage and does not require credentials).
3. Use a `RuleDictionary` object to load a particular dictionary, as shown in Example 4–3, which loads the `CarRentalxml` dictionary into the object named `dict`. The `CarRentalxml` dictionary was previously created using Rule Author.

***Example 4–3 Loading a Dictionary with Rules SDK (XML)***

```
String repoPath = "dict" + fs + "CarxmlRepository";
final String jarstoreKey = "oracle.rules.sdk.store.jar";
RepositoryType jarType =
   RepositoryManager.getRegisteredRepositoryType( jarstoreKey );
RuleRepository repo = RepositoryManager.createRuleRepositoryInstance( jarType );
RepositoryContext jarCtx = new RepositoryContext();
jarCtx.setProperty( oracle.rules.sdk.store.jar.Constants.I_PATH_BASE, repoPath );
repo.init( jarCtx );

RuleDictionary dict = repo.loadDictionary( "CarRentalxml", "HowToxml" );
```

If you want to load a WebDAV repository instead of a file repository as shown in Example 4–3, you should use `getWebDAVRepository`. An example of this is shown in `TestXML.java` in the $*HowToDir*`/src/carrental` directory.

### 4.7.4 Loading a Ruleset and Generating RL Language for Data Model and Rule Set

After loading a dictionary, you can use the Rules SDK to generate an RL Language program. This step is required since a dictionary stores a data model and rule sets using an intermediate XML format. The `RuleDictionary` provides methods to access a data model and a rule set and perform the mapping from the intermediate XML format. This mapping produces the RL Language data program.

When you generate rules using Rule Author, each rule set specifies two components, a data model which is global and applies for all the rule sets in a dictionary, and the set of rules associated with a rule set.

Example 4–4 shows the code that generates the RL Language code for a rule set and for the associated data model.

***Example 4–4 Generating Oracle Business Rules RL Language***

```
String rsname = "vehicleRent";
String dmrl = dict.dataModelRL();
```

```
String rsrl = dict.ruleSetRL( rsname );
```

### 4.7.5 Initializing and Executing a Rule Session

After you generate an RL Language program that include rules and a data model, you are ready to work with a rule session. A rule session initializes the Rules Engine and maintains the state of the Rules Engine across a number of rule executions.

Example 4–5 shows the code that creates a `RuleSession` object and executes an RL Language program.

The `executeRuleset()` method tells the Rules Engine to interpret the specified RL Language program.

> **Note:** The order of the `executeRuleset()` calls is important. You need to execute the data model RL Language program before the rule set RL Language program. The data model contains global information that is required when the associated rule set executes.

***Example 4–5 Initializing and Executing a Rule Session with Rules SDK (XML)***

```
RuleSession session = new RuleSession();
session.executeRuleset( dmrl );
session.executeRuleset( rsrl );

session.callFunction( "reset" );
session.callFunction( "clearRulesetStack"  );
session.callFunctionWithArgument( "pushRuleset", rsname );
```

After the data model and the rule set are loaded and the rule session is ready to run the rule set against the facts that you assert for the rule session.

### 4.7.6 Asserting XML Data from Within a Rule Session

Before running a rule session you need to first unmarshall the XML document containing the XML data and then assert the facts from the XML document. Section 4.7.2 shows you how to unmarshall the XML document.

To assert facts from an XML document, use the `session.callFunctionWithArgument()` method with the `assertXPath` function as an argument.

Example 4–6 shows sample code that uses `assertXPath` to assert XML facts into a rule session.

The `callFunctionWithArgumentList` method requires a function name argument and a List argument. The List argument `argList` includes the following three arguments:

1. The first argument for `assertXPath` is the JAXB generated package name, for this example, `generated`.
2. The second argument for `assertXPath` is the root object for the unmarshalled XML document. For this example the unmarshalled object reference is the `root` object.
3. The third argument for `assertXPath` is the XPath expression to assert, for this example the `"//*"` asserts the entire XML tree into the rule session named `session`.

***Example 4–6 Asserting an XML Document***

```
List argList = new ArrayList(3);
argList.add( "generated" );
argList.add( root );
argList.add( "//*" );
session.callFunctionWithArgumentList( "assertXPath", argList );
```

**See Also:** "Creating a JAXB Context and Unmarshalling the XML Document" on page 4-24

### 4.7.7 Using the Run Function with a Rule Session

Example 4–7 shows the code that runs a rule session.

***Example 4–7 Running a Rules Engine Session***

```
session.callFunction( "run");
```

## 4.8 Running the XML Car Rental Sample Using the Test Program

The *$HowToDir*/lib directory includes TestXML.jar, a ready-to-run Oracle Business Rules Java application that uses the CarRentalxml dictionary. If you change the dictionary name and you need to modify TestXML.java, the source is available in the directory *$HowToDir*/src. The Readme.txt file in this directory includes instructions for setting the environment variables required to run the test program.

Where *$HowToDir* is the directory where you installed the Oracle Business Rules XML How-To.

Example 4–8 shows output from running TestXML.

***Example 4–8 Sample Run of Car Rental Program (XML)***

```
java carrental.TestXML
Rental declined Qun Under age, age is: 15
```

Note that not all facts produce output or fire a rule. The example shows output only for the asserted fact that matches the UnderAge rule.

# 5 Using JSR-94

This chapter covers the following:

- Oracle Business Rules with JSR-94 Rule Execution Sets
- Using the JSR-94 Interface with Oracle Business Rules

## 5.1 Oracle Business Rules with JSR-94 Rule Execution Sets

To use JSR-94 with rules created either with Rule Author or in RL Language text, you need to map the rules to a JSR-94 rule execution set. A JSR-94 rule execution set (rule execution set) is a collection of rules that are intended to be executed together. You also need to register a rule execution set before running the rule execution set. A registration associates a rule execution set with a URI; using the URI you can create a JSR-94 rule session.

This section covers the following topics:

- Creating a JSR-94 Rule Execution Set from Rule Sets in a File Repository
- Creating a JSR-94 Rule Execution Set from a WebDAV Repository
- Creating a Rule Execution Set from Oracle Business Rules RL Language Text
- Creating a Rule Execution Set from RL Text Specified in a URL
- Creating Rule Execution Sets with Rule Sets from Multiple Sources

---

**Note:** Using Oracle Business Rules, a JSR-94 rule execution set registration is not persistent. Thus, you need to register a rule execution set programmatically using a JSR-94 `RuleExecutionSetProvider` interface.

---

### 5.1.1 Creating a JSR-94 Rule Execution Set from Rule Sets in a File Repository

You can save rules created with Rule Author in a dictionary using the dictionary storage plug-in. To use JSR-94 with rules created with Rule Author, you need to map a Rule Author dictionary and its contents to a JSR-94 rule execution set.

Perform the following steps to use a Rule Author dictionary with JSR-94:

1. Specify Rule Author dictionary mapping information in an XML document. Table 5–1 shows the mapping elements required to construct a rule execution set. Example 5–1 shows a sample XML mapping file.

2. You then use the XML document with the JSR-94 administration APIs to create a rule execution set. The resulting rule execution set is registered with a JSR-94 runtime (using a `RuleAdministration` instance).

***Table 5–1 File Repository XML Mapping Elements for JSR-94***

| Element | Description |
|---|---|
| `<repository-location>` | The file repository path – the path may be absolute or relative to the current directory at the time of execution. |
| `<dictionary-name>` | The dictionary name. |
| `<dictionary-version>` | The dictionary version. |
| `<ruleset-list>` | A list of Rule Author rule sets to extract from the dictionary in the order in which they should be interpreted so that any inter-dependencies are resolved.<br>Note: the rule set associated with data model is not included in the `<ruleset-list>` element. The JSR-94 implementation loads the data model rule set into the Rules Engine before any rule sets listed in this element. |
| `<ruleset-stack>` | Specifies a list of rule sets that make up the initial rule set stack. The order specified for the of rule sets in the list is from the top of the stack to the bottom of the stack. |

***Example 5–1 JSR-94 XML Mapping File for a File Repository***

```
<rule-execution-set xmlns="http://xmlns.oracle.com/rules/jsr94/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0">
   <name>CarRentalDemo</name>
   <description>The Car Rental Demo</description>
   <rule-source>
      <file-repository>
         <repository-location>dict/CarRepository</repository-location>
         <dictionary-name>CarRental</dictionary-name>
         <dictionary-version>HowTo</dictionary-version>
         <ruleset-list>
            <ruleset-name>vehicleRent</ruleset-name>
         </ruleset-list>
      </file-repository>
   </rule-source>
   <ruleset-stack>
      <ruleset-name>vehicleRent</ruleset-name>
   </ruleset-stack>
</rule-execution-set>
```

**See Also:** The XSD file in *`$ORACLE_HOME`*`/rules/lib/jsr94_obr.jar` at `oracle/rules/jsr94/admin/jsr94-runtime-configuration-1.0.xsd`.

### 5.1.2 Creating a JSR-94 Rule Execution Set from a WebDAV Repository

You can save rules created with Rule Author in a WebDAV repository using the dictionary storage plug-in. To use JSR-94 with rules stored in a WebDAV repository, you need to map one or more rule sets from a WebDAV repository to a JSR-94 rule execution set.

Perform the following steps to use rules stored in a file repository with JSR-94:

1. Specify WebDAV repository mapping information in an XML document. Table 5–2 shows the mapping elements required to construct a rule execution set. Example 5–2 shows a sample XML mapping file.
2. You then use the XML document with the JSR-94 administration APIs to create a rule execution set. The resulting rule execution set is registered with a JSR-94 runtime (using a RuleAdministration instance).

*Table 5–2 WebDAV Repository XML Mapping Elements for JSR-94*

| Element | Description |
|---|---|
| `<repository-url>` | The URL for the WebDAV repository. |
| `<proxy-host>` | The name of the proxy host if a proxy is present. This is an optional element. |
| `<proxy-port>` | The proxy port if a proxy is present. This is an optional element. |
| `<dictionary-name>` | The dictionary name. |
| `<dictionary-version>` | The dictionary version. |
| `<ruleset-list>` | A list of Rule Author rule sets to extract from the dictionary in the order in which they should be interpreted so that any inter-dependencies are resolved.<br>Note: the rule set associated with data model is not included in the `<ruleset-list>` element. The JSR-94 implementation loads the data model rule set into the Rules Engine before any rule sets listed in this element. |
| `<ruleset-stack>` | Specifies a list of rule sets that make up the initial rule set stack. The order specified for the of rule sets in the list is from the top of the stack to the bottom of the stack. |

*Example 5–2 JSR-94 Mapping File for a WebDAV Repository*

```
<rule-execution-set xmlns="http://xmlns.oracle.com/rules/jsr94/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0">
   <name>CarRentalDemo</name>
   <description>The Car Rental Demo</description>
   <rule-source>
      <webdav-repository>
         <repository-url>
            http://www.some_server.com/rules_repository
         </repository-url>
         <dictionary-name>CarRental</dictionary-name>
         <dictionary-version>HowTo</dictionary-version>
         <ruleset-list>
            <ruleset-name>vehicleRent</ruleset-name>
         </ruleset-list>
      </webdav-repository>
   </rule-source>
   <ruleset-stack>
      <ruleset-name>vehicleRent</ruleset-name>
   </ruleset-stack>
</rule-execution-set>
```

### 5.1.3 Creating a Rule Execution Set from Oracle Business Rules RL Language Text

You can use JSR-94 with RL Language rule sets saved as text, where the RL Language text is directly included in the rule execution set.

Perform the following steps to use RL Language specified rules with JSR-94:

1. Specify the RL Language mapping information in an XML document. Table 5–3 shows the mapping elements required to construct a rule execution set. Example 5–3 shows a sample XML document for mapping RL Language text to a JSR-94 rule execution set.
2. You then use the XML document with the JSR-94 administration APIs to create a rule execution set. The resulting rule execution set is registered with a JSR-94 runtime (using a `RuleAdministration` instance).

***Table 5–3 Oracle Business Rules RL Language Text XML Mapping Elements for JSR-94***

| Element | Description |
|---|---|
| `<rule-source>` | Includes an `<rl-text>` tag containing explicit RL Language text containing an Oracle Business Rules rule set. Multiple `<rule-source>` tags can be used to specify multiple rule sets (specified in the order in which they are interpreted). |
| `<ruleset-stack>` | Specifies a list of rule sets that make up the initial rule set stack. The order of the rule sets in the list is from the top of the stack to the bottom of the stack. |

---

**Note:** In the `<rl-text>` element the contents must escape XML predefined entities. This includes the characters, '&', '>', '<', '"', and '\'.

---

***Example 5–3 XML Mapping File for Rule Sets Defined in an RL Program***

```
<rule-execution-set xmlns="http://xmlns.oracle.com/rules/jsr94/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0">
   <name>CarRentalDemo</name>
   <description>The Car Rental Demo</description>
   <rule-source>
      <rl-text>
         ruleset DM {
            fact class carrental.Driver {
               hide property ableToDrive, driverLicNum, licIssueDate, licenceType,
               llicIssueDate, numPreAccidents, numPreConvictions,
               numYearsSinceLicIssued, vehicleType;
            };

               final String DeclineMessage = &quot;Rental declined &quot;;

               public class Decision  supports xpath {
                  public String driverName;
                  public String type;
                  public String message;
               }

               function assertXPath(String package,
                                    java.lang.Object element, String xpath) {
                  //RL literal statement
                  main.assertXPath( package, element, xpath );
               }

               function println(String message) {
                  //RL literal statement
                  main.println(message);
               }
```

```
                function showDecision(DM.Decision decision) {
                   //RL literal statement
                   DM.println( &quot;Rental decision is &quot; + decision.type +
                               &quot; for driver &quot; + decision.driverName +
                               &quot; for reason &quot; + decision.message);
                }
             }
          </rl-text>
       </rule-source>
       <rule-source>
          <rl-text>
             ruleset vehicleRent {
                rule UnderAge {
                   priority = 0;
                   if ((fact carrental.Driver v0_Driver &amp;&amp;
                      (v0_Driver.age &lt; 19))) {
                      DM.println( &quot;Rental declined: &quot; + v0_Driver.name +
                                  &quot; Under age, age is: &quot; + v0_Driver.age);
                      retract(v0_Driver);
                   }
                }
             }
          </rl-text>
       </rule-source>
       <ruleset-stack>
          <ruleset-name>vehicleRent</ruleset-name>
       </ruleset-stack>
</rule-execution-set>
```

**See Also:** "Using the Extended createRuleExecutionSet to Create a Rule Execution Set" on page 5-8 for information on JSR-94 extensions that assist you in including RL Language text.

### 5.1.4 Creating a Rule Execution Set from RL Text Specified in a URL

You can use JSR-94 with RL Language rule sets specified using a URL.

To use RL Language specified rules with JSR-94, do the following:

1. Specify the RL Language mapping information in an XML document. Table 5–4 shows the mapping elements required to construct a rule execution set. Example 5–4 shows a sample XML document for mapping RL Language text to a JSR-94 rule execution set.
2. You then use the XML document with the JSR-94 administration APIs to create a rule execution set. The resulting rule execution set is registered with a JSR-94 runtime (using a `RuleAdministration` instance).

***Table 5–4 Oracle Business Rules RL Language URL XML Mapping Elements for JSR-94***

| Element | Description |
|---|---|
| <rule-source> | Includes an `<rl-url>` tag containing a URL that specifies the location of RL Language text. Multiple `<rule-source>` tags can be used to specify multiple rule sets (in the order in which they are interpreted). |
| `<ruleset-stack>` | Specifies a list of rule sets that make up the initial rule set stack. The order of the rule sets in the list is from the top of the stack to the bottom of the stack. |

***Example 5–4 XMP Mapping File for Rule Sets Defined in a URL***

```
<?xml version="1.0" encoding="UTF-8"?>
<rule-execution-set xmlns="http://xmlns.oracle.com/rules/jsr94/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0">
   <name>CarRentalDemo</name>
   <description>The Car Rental Demo</description>
   <rule-source>
      <rl-url>
         file:rl/DM.r1
      </rl-url>
   </rule-source>
   <rule-source>
      <rl-url>
         file:r1/VehicleRent.r1
      </rl-url>
   </rule-source>
   <ruleset-stack>
      <ruleset-name>vehicleRent</ruleset-name>
   </ruleset-stack>
</rule-execution-set>
```

**See Also:** "Using the Extended createRuleExecutionSet to Create a Rule Execution Set" on page 5-8 for information on JSR-94 extensions that assist you in specifying a URL.

### 5.1.5 Creating Rule Execution Sets with Rule Sets from Multiple Sources

A rule execution set may contain rules that are derived from multiple sources and the sources may be a mix of Rule Author defined rule sets and RL Language rule sets. In this case, the XML element `<rule-execution-set>` set contains multiple `<rule-source>` elements, one for each source of rules. You need to list each `<rule-source>` in the order in which they are to be interpreted in the rules engine.

**Note:** For this Oracle Business Rules release, a JSR-94 rule execution set can only reference one Rule Author dictionary.

## 5.2 Using the JSR-94 Interface with Oracle Business Rules

This section describes some Oracle Business Rules specific details for JSR-94 interfaces. This section covers the following topics:

- Creating a Rule Execution Set with CreateRuleExecutionSet
- Creating a Rule Session with createRuleSession
- Working with JSR-94 Metadata
- Using Oracle Business Rules JSR-94 Extensions

### 5.2.1 Creating a Rule Execution Set with CreateRuleExecutionSet

The `RuleExecutionSetProvider` and `LocalRuleExecutionSetProvider` interfaces in `javax.rules.admin` include the `createRuleExecutionSet` to create a `RuleExecutionSet`.

For the remaining `createRuleExecutionSet` methods, the first argument is interpreted as shown in Table 5–5.

**Table 5–5 First Argument Types for createRuleExecutionSet Method**

| Argument | Description |
|---|---|
| `org.w3c.dom.Element` | Specifies an instance of the `<rule-execution-set>` element from the configuration schema. |
| `java.lang.String` | Specifies a URL that specifies the location of an XML document that is an instance of the `<rule-execution-set>` element from the configuration schema. |
| `java.io.InputStream` | Specifies an input stream that is used to read an XML document that is an instance of the `<rule-execution-set>` element from the configuration schema. |
| `java.io.Reader` | Specifies a character reader that is used to read an XML document that is an instance of the `<rule-execution-set>` element from the configuration schema. |

> **Note:** JSR-94 also includes `createRuleExecutionSet` methods that take a `java.lang.Object` argument, which is intended to be an abstract syntax tree for the rule execution set. In this release of Oracle Business Rules, using the method with this argument is not supported. Invoking these methods with a `java.lang.Object` argument gives a `RuleExecutionSetCreateException`.

The second argument to the `createRuleExecutionSet` methods is a `java.util.Map` of vendor specific properties. The properties in Table 5–6 are valid for the Oracle JSR-94 implementation.

**Table 5–6 createRuleExceptionSet Oracle Specific Properties**

| Property Key | Property Value |
|---|---|
| `oracle.rules.jsr94.sensitiveDataCallback` | This property is set when authentication is required by the selected repository such as a WebDAV server that is configured to require authentication. The property value must be an implementation of the `oracle.rules.sdk.repository.SensitiveDataCallback` interface. |

## 5.2.2 Creating a Rule Session with createRuleSession

Clients create a JSR-94 rule session using the `createRuleSession` method in the `RuleRuntime` class. This method takes a `java.util.Map` argument of vendor specific properties. This argument can be used to pass in any of the properties defined for the Oracle Business Rules `oracle.rules.rl.RuleSession`. If a rule execution set contains URL or repository rule sources, the rules from those sources are fetched on the creation of each new `RuleSession`.

## 5.2.3 Working with JSR-94 Metadata

JSR-94 allows for metadata for rule execution sets and rules within a rule execution set. The Oracle Business Rules implementation does not add any additional metadata beyond what is in the JSR-94 specification. The rule execution set description is an optional item and thus may not be present. If it is not present, the empty string is returned. For rules, only the rule name is available and the description is initialized with an empty string.

### 5.2.4 Using Oracle Business Rules JSR-94 Extensions

This section covers the following extensions provided in the JSR-94 implementation classes.

- Using the Extended createRuleExecutionSet to Create a Rule Execution Set
- Invoking an RL Function in JSR-94

#### 5.2.4.1 Using the Extended createRuleExecutionSet to Create a Rule Execution Set

Oracle Business Rules provides a helper function to facilitate creating the XML control file required as input to create a `RuleExecutionSet`.

The helper method `createRuleExecutionSet` is available in the `RLLocalRuleExecutionSetProvider` class. The `createRuleExecutionSet` method has the following signature:

```
RuleExecutionSet createRuleExecutionSet(String name,
                                        String description,
                                        RuleSource[] sources,
                                        String[] rulesetStack,
                                        Map properties)
```

Table 5–7 describes the `createRuleExecutionSet` arguments.

*Table 5–7 createRuleExecutionSet Arguments*

| Argument | Description |
|---|---|
| `name` | Specifies the name of the rule execution set. |
| `description` | Specifies the description of the rule execution set. |
| `sources` | Specifies an array of specifications for the sources of rules. In this release, four types of sources are supported: RL Language text, a URL to RL Language text, a file repository (.jar file), and a WebDAV repository. `RuleSource` is an interface that the classes `RLTextSource` (RL Language text), `RLUrlSource` (RL Language URL), `JarRepositorySource` (file repository), and `WebDAVRepositorySource` (WebDAV repository) implement.<br><br>For more information, see the `oracle.rules.jsr94.admin` package in *Oracle Business Rules Java API Reference*. |
| `rulesetstack` | Specifies the initial contents of the RL Language rule set stack to be set prior to each time the rules are executed. The contents of the array should be ordered from the top of stack (0th element) to the bottom of stack (last element). |
| `properties` | Oracle specific properties. See Table 5–6. |

#### 5.2.4.2 Invoking an RL Function in JSR-94

In a stateful interaction with a JSR-94 rule session, a user may want the ability to invoke an arbitrary RL Language function. The class that implements the JSR-94 `StatefulRuleSession` interface provides access to the `callFunction` methods in the `oracle.rules.rl.RuleSession` class.

Example 5–5 shows how you can to invoke an RL Language function with no arguments in a JSR-94 `StatefulRuleSession`.

*Example 5–5 Using CallFunction with a StatefulRuleSession*

```
import javax.rules.*;
 ...
```

```
StatefulRuleSession session;
...
((oracle.rules.jsr94.RLStatefulRuleSession) session).callFunction("myFunction");
```

# 6 Using Oracle Business Rules SDK

Oracle Business Rules SDK (Rules SDK) provides APIs that a developer can use to write customized applications that access, create, or modify rules and data models (and all the information stored in an Oracle Business Rules dictionary). Using Rules SDK APIs, you can create, modify, and access dictionary data using well defined interfaces and you can use the APIs to build customized rules enabled applications.

You can use the Rules SDK APIs in a rule enabled application to access existing rules and then run the rules engine, or in an application that you write to access, create, or edit rules and data model information.

This chapter introduces the Oracle Business Rules SDK APIs.

This chapter covers the following topics:

- Rules SDK building blocks
- Working with a Repository and a Dictionary
- Working with a Data Model
- Creating a DataModel
- Using RuleSets and Creating and Modifying Rules

## 6.1 Rules SDK building blocks

The top level Rules SDK package, `oracle.rules.sdk,` includes the following packages:

- `oracle.rules.sdk.repository`
- `oracle.rules.sdk.dictionary`
- `oracle.rules.sdk.editor.datamodel`
- `oracle.rules.sdk.editor.ruleset`
- `oracle.rules.sdk.exception`

The Rules SDK interface follows the Java Bean model and includes getters and setters for each bean property. For example, `setName("somevalue")` sets the `name` property of the individual instance.

In addition to bean interfaces, the Rules SDK provides a hash get and put style interface. The bean interfaces are generally useful, but there is a least one GUI framework in which the HashMap style is necessary.

## 6.2 Working with a Repository and a Dictionary

Oracle Business Rules dictionaries are stored in a repository. Prior to accessing a dictionary, access to its repository must be established. This requires specifying the type of repository to access and the initialization parameters required by that specific repository type. As shipped, Rule Author supports a WebDAV (Web Distributed Authoring and Versioning) repository and a file repository.

Table 6–1 shows the initialization parameter keys for a WebDAV repository. The repository type key is `oracle.rules.sdk.store.webdav`.

***Table 6–1 WebDAV Repository Type Parameter Initialization Keys***

| Parameter | Key | Description |
|---|---|---|
| URL | `oracle.rules.sdk.store.webdav.url` | The URL for the desired WebDAV rule repository. This parameter is required. |
| Proxy Host | `oracle.rules.sdk.store.webdav.proxyHost` | The host name of the proxy server. This is only required if you have a proxy server between the server on which Rule Author is running and the WebDAV server. |
| Proxy Port | `oracle.rules.sdk.store.webdav.proxyPort` | The port to use for the proxy server. This is only required if you have a proxy server between the server on which Rule Author is running and the WebDAV server. |

Table 6–2 shows the initialization parameter keys for a file repository. The repository type key is `oracle.rules.sdk.store.jar`.

***Table 6–2 File Repository Type Parameter Initialization Key***

| Parameter | Key | Description |
|---|---|---|
| File Path | `oracle.rules.sdk.store.webdav.path` | The path to the file that contains the rule repository. This parameter is required. |

### 6.2.1 Establishing Contact with a WebDAV Repository

Example 6–1 shows how to establish access to a WebDAV repository.

***Example 6–1 Establishing Access to a WebDAV Repository***

```
String url;     // the URL for the WebDAV repository
Locale locale;  // the desired Locale

// The following code assumes that the url and locale have been set appropriately
RepositoryType rt =
   RepositoryManager.getRegisteredRepositoryType("oracle.rules.sdk.store.webdav");
RuleRepository repos = RepositoryManager.createRuleRepositoryInstance(rt);
RepositoryContext rc = new RepositoryContext();
rc.setLocale(locale);
rc.setProperty("oracle.rules.sdk.store.webdav.url", url);
repos.init(rc);
```

If the WebDAV repository has been configured to require authentication, then the following must be performed:

- Configure a wallet with the required user name(s) and password(s).
- Create an instance of the `oracle.rules.sdk.callbacks.WalletCallback` class and set it in the `RepositoryContext` prior to calling the `init` method.

In Example 6–2, `/wallets/rules_wallet` is the path to the wallet configured with the credentials for WebDAV authentication:

***Example 6–2 Configuring a Wallet for Authentication***

```
WalletCallback callback = new WalletCallback("/wallets/rules_wallet", null);
rc.setSensitiveDataCallback(callback);
```

### 6.2.2 Establishing Contact with a File Repository

Example 6–3 shows how to establish access to a file repository.

***Example 6–3 Establishing Access to a File Repository***

```
String path;     // the path to the file repository
Locale locale;  // the desired Locale

// The following code assumes that the path and locale have been set appropriately
RepositoryType rt =
   RepositoryManager.getRegisteredRepositoryType("oracle.rules.sdk.store.jar");
RuleRepository repos = RepositoryManager.createRuleRepositoryInstance(rt);
RepositoryContext rc = new RepositoryContext();
rc.setLocale(locale);
rc.setProperty("oracle.rules.sdk.store.jar.path", path);
repos.init(rc);
```

### 6.2.3 Loading a Dictionary

Now, a dictionary may be loaded either by specifying the dictionary name, in which case, the default version of the dictionary is loaded with:

```
RuleDictionary dictionary = repos.loadDictionary(dictionaryName);
```

or by specifying both the dictionary name and version with:

```
RuleDictionary dictionary = repos.loadDictionary(dictionaryName,
                                                 dictionaryVersion);
```

These examples are applicable to both file and WebDAV repositories.

## 6.3 Working with a Data Model

The Rules SDK data model contains the fact types, internal variables, constraints, and functions that you use to create rules. The fact types from the data model can be reasoned on in corresponding rules. Oracle Business Rules variables contain information that rules share. Oracle Business Rules functions provide for logic reuse for rules. Constraints limit the set of valid values for rule customization.

> **Note:** To import existing Java classes or XML schemas, the Rule Author application must be used. After the classes or schemas are imported with Rule Author, the repository may be used by the SDK. Future versions of the SDK will have extensions to allow for Java classes and XML schemas to be imported directly.

After you use a repository to create or open a `RuleDictionary`, you can use the Rules SDK to create a data model in the dictionary. A `RuleDictionary` can access the internal data structures necessary to create a `DataModel` instance.

The data model shown in the examples is this chapter includes the Java FactTypes that are imported from a sample package named `email`. The `email` package was imported using Rule Author. The classes and properties shown in Table 6–3 were populated by importing `email.jar`.

***Table 6–3 Sample email Package Classes***

| Name | Description | Type |
|---|---|---|
| `email.ElectronicMessage` | Represents the occurrence of a message | Java FactType |
| `email.EmailAddress` | Represents an email address | Java FactType |
| `email.EmailAddressList` | Represents a list of email addresses | Java FactType |

In the data model for the spam processing example, using the `email` package, the data model supports inferencing by creating an `RLFact` named `SpamFound`. To contain a global count, we create a variable named `spamCounter`, and the constant `String` variable indicates spam. A function named `killSpam` provides an action when the rules detect that an email message is spam. Table 6–4 shows these data model components.

***Table 6–4 Sample Data Model Types for Handling email Package***

| Name | Description | Type |
|---|---|---|
| `SpamFound` | Asserted when email message is determined to be spam | RL FactType |
| `spamCounter` | Accumulates count of spam messages | variable |
| `SpecialOffer_CONST` | Constant containing the `String` "Special Offer" | constant variable |
| `fKillSpam` | Called when spam found to delete spam | RL Function |

## 6.3.1 Creating a DataModel

After you create and open a `RuleDictionary`, you can use the Rules SDK to create an `editor.DataModel`. The `RuleDictionary` can access the internal data structures necessary to create a `DataModel` instance.

For example,

```
eDM = new oracle.rules.sdk.editor.datamodel.DataModel(m_dict);
```

The basis of the DataModel type system is the FactType. A FactType is defined as a primitive, Java, XML, or RL FactType. A Primitive FactType is fixed, and includes Java primitives (for example: `String`, `int`, or `double`). The Rules SDK automatically creates primitive types when you create a `RuleDictionary`. You create Java and XML FactTypes when you import classes from jar file, a class, or a schema file. You can create RL FactTypes directly using the Rules SDK. The Java, XML, and RL FactTypes define classes and may have associated properties, which represent the JavaBean defined properties, and methods.

## 6.3.2 Creating DataModel Components

You create each part of the data model using the appropriate `ModelComponentTable`. The sequence required to create a new instance, `Function`, `FactType`, `Variable`, or `Constraint`, is the same:

- Instantiate the appropriate table in the data model.
- Invoke the table instance `add()` method.

Example 6–4 shows how you create a Function instance:

***Example 6–4 Creating a Function Instance***

```
FunctionTable ft = eDM.getFunctionTable();
Function fKillSpam = ft.add();
```

To import existing Java classes or XML schemas, the Rule Author application must be used. After the classes or schemas are imported with Rule Author, the repository may be used by the SDK. Future versions of the SDK will have extensions to allow for Java classes and XML schemas to be imported directly.

### 6.3.3 Creating a Function Argument List

Using the Rules SDK you create argument lists for functions using a `FormalParameterTable`. Each `FormalParameter` entry represents a parameter. The first parameter is the first (zero) entry in the `FormalParameterTable`, the second parameter is the second Entry, and so on. Variables may be constants, which may not be assigned a value after the original initialization. The `setFinal()` method controls the constant behavior. Each `FormalParameter` has a type and a name. The type of a formal parameter is selected from the available FactTypes.

The code in Example 6–5 creates a `FormalParameterTable` for the `email` sample.

***Example 6–5 Creating a FormalParameterTable***

```
// define the parms of the function
// basically just an instance of ElectronicMessage
// and a String to explain what triggered this
FormalParameterTable fKillSpamParmTable = fKillSpam.getFormalParameterTable();
FormalParameter fp1 = fKillSpamParmTable.add();
FormalParameter fp2 = fKillSpamParmTable.add();
fp1.setName("emsg");
fp1.setAlias("Email Message");

// use the alias for the email.ElectronicMessage fact type
// will be in the getType_Options list
fp1.setType("email.ElectronicMessage");
fp2.setName("reason");
fp2.setAlias("reason");

 // use the primitive type for String
 // will be in the getType_Options list
fp2.setType("String");
```

### 6.3.4 Creating an Initializing Expression

Using the Rules SDK, variables contain state internal to a RuleSet. Each variable must have a type and an initializing Expression. The variable type of is chosen from the list of possible FactTypes (all FactTypes defined in the DataModel).

There are two types of expressions:

- `Expression`

  This type of expression must follow the form:

  *`operand operator operand`*

  The operands and operators available for an `Expression` are more limited than than those for an `AdvancedExpression`. For example, an `Expression` does not allow an operand to be a function requiring parameters.

- `AdvancedExpression`

  This type of expression allows for the full range of operands and operators.

It is recommended that you use an `AdvancedExpression` to initialize an expression. Example 6–6 shows how to set a variable's initial value.

***Example 6–6 Setting a Variable's Initial Value***

```
InitialValue iv = var.getValue();
AdvancedExpression adv = iv.getAdvancedExpression();
adv.insert(0, "\"FIXEDVALUE\"");
```

### 6.3.5 Creating RL Function Bodies

RL Function bodies are composed of strings of RL Language. They may refer to any of the Function parameters, or any global Variable. Enter function bodies as a String that must be syntactically correct RL Language (see Example 6–7).

***Example 6–7 Creating a Function Body***

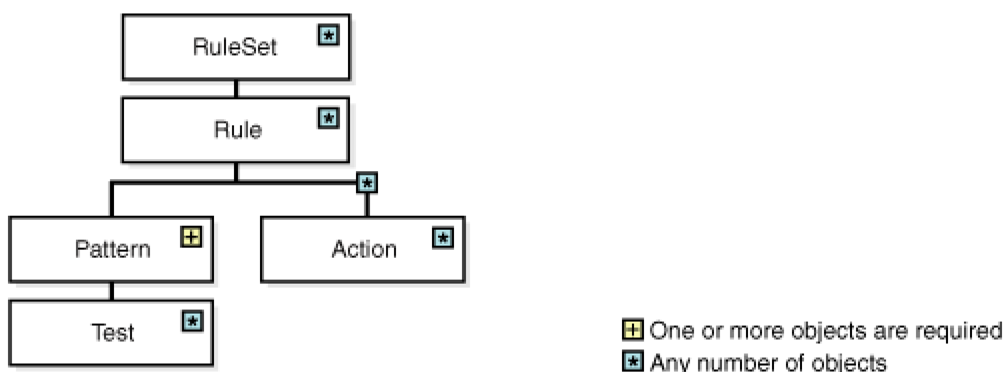
```
//set the body of the function
// in this case just pretty print a message
fKillSpam.setBody(" println(\" email from: \" + emsg.getSender() + \"  because \"
+ reason)" );
```

## 6.4 Using RuleSets and Creating and Modifying Rules

Using the Rules SDK, a rule is a conditional expression, referred to as a condition, and a set of actions that execute if the condition evaluates to true. A rule condition is composed of a set of patterns. A pattern delineates the match type and includes tests against that type and other types that appear in preceding pattern (order has meaning). Rule actions can be calls, assignments, retractions, and assertions.

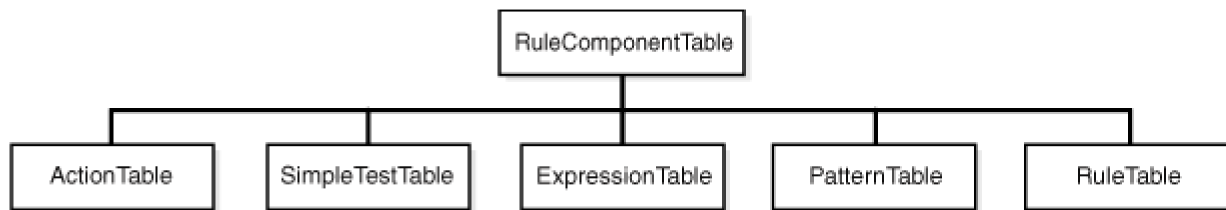
Figure 6–1 shows the general container hierarchy for a RuleSet.

***Figure 6–1 Rules SDK RuleSet Container Hierarchy***




The Rules SDK provides classes that represent each of these objects, all classes descend from `RuleComponent`. For collections, for example rules in a ruleset, patterns in a rule, or actions in a rule, the Rules SDK provides a class that is descended from `RuleComponentTable` to manage a specific collection (see Figure 6–2). The `RuleComponentTable` sub-class provides a specialized `add()` method for the particular sub-class that the table represents.

***Figure 6–2 Rules SDK RuleComponents***




The Rules SDK components describing the XML schemas, Java Classes, RL Global variables, and RL Functions are located in the data model (stored in the dictionary). Typically, `RuleComponent` instances refer to data model entities.

## 6.4.1 Creating a RuleSet

Generally, the sequence you use to create a `RuleComponent` is:

- Create a `RuleSet` by use of a `RuleSet` constructor.
- Create any children of the `RuleSet` by `add()` methods on the appropriate table (`PatternTable`, `ActionTable`, `ExpressionTable`, `SimpleTestTable`).

The Rules SDK API provides classes that represent collections in a `RuleSet`. Each of these objects descend from `RuleComponent`. For collections, for example rules in a ruleset, or patterns in a rule or actions in a rule, the Rules SDK provides classes that are descended from `RuleComponentTable` class to manage a specific collection. The `RuleComponentTable` sub-class provides a specialized `add()` method for the particular sub-class represented by the table.

The code in Example 6–8 creates a `RuleSet` for the email sample,

***Example 6–8 Creating a RuleSet***

```
oracle.rules.sdk.editor.ruleset.RuleSet rs = null;
  try
    {
      rs = new oracle.rules.sdk.editor.ruleset.RuleSet(m_dict);
      m_curBean = rs;
      rs.setName("SpamRuleSet");
    }
    catch (Exception e)
    {
      System.out.println(" create RuleSet FAILED");
      addException(e);
      return;
    }
```

## 6.4.2 Adding a Rule to a Ruleset

The Rules SDK API provides classes that represent the objects in a `RuleSet`. Each of these objects descend from `RuleComponent`. For collections, for example rules in a ruleset, or patterns in a rule or actions in a rule, the Rules SDK provides classes that are descended from `RuleComponentTable` class to manage a specific collection. The `RuleComponentTable` sub-class provides a specialized `add()` method for the particular sub-class represented by the table.

There are several classes that are not parts of collections. To access these classes use the the parent bean with the getter interface. For example, acquire the

`AdvancedExpression` by invoking `getAdvancedExpression()` on the correct `Pattern` instance.

The code in Example 6–9 shows how to add a rule to a table in RuleSet.

***Example 6–9 Adding a Rule to a Table in a RuleSet***

```
//add rule to the table
oracle.rules.sdk.editor.ruleset.Rule r = rs.getRuleTable().add();
r.setName("DetectSpamRule");
```

## 6.4.3 Adding a Pattern to a Rule

The Rules SDK API provides the `Pattern` class that represents a pattern. The `getPatternTable` sub-class provides a specialized `add()` method for adding a pattern object, as shown in Example 6–10.

***Example 6–10 Adding a Pattern to a Rule***

```
//add pattern to the rule
oracle.rules.sdk.editor.ruleset.Pattern p = r.getPatternTable().add();

//set pattern
p.setVariable("xx");
p.setFactType( "email.ElectronicMessage");
p.setTestForm("Advanced");
```

The `TestForm` property defines the type of test associated with the pattern.

## 6.4.4 Adding a Test to a Pattern

Every `Pattern` may have Tests associated with the `Pattern`. Tests may take the form of `SimpleTest` or `AdvancedExpression`. A `Pattern` may have an unlimited number of Tests.

The Rules SDK "ANDs" Tests together when generating the RL Language. For example, the test used for the email example requires a function with parameters. This requires an `AdvancedExpression` (see Example 6–11).

***Example 6–11 Adding a Test to a Pattern***

```
AdvancedExpression adv = (AdvancedExpression)p.get("AdvancedExpression");

// FUNCTION and Variable complex expression
adv.put("Function", "containsString");
adv.insert(0, adv.getFunctionDescription());

//System.out.println(" function is: <<" + adv.getFunctionDescription() + ">>");
//set the function parms
// normally the cursor position is set by user input actions
adv.setVariable("SPECIAL OFFER");
adv.replace(17, 57, adv.getVariable() );
adv.insert( ((String)adv.getValue()).length() , "," );
adv.setVariable("message.subject");
adv.insert( ((String)adv.getValue()).length() + 1,
adv.getVariable() );
adv.insert( ((String)adv.getValue()).length() , " )" );
//since boolean, this is a single operator no need for anything else
```

If the test is of the form:

*operand operator operand*

where neither operand is a function requiring parameters, then a `SimpleTest` may be used. For example, if the pattern variable name is "emsg" and the test is "emsg.sender == david@fun.com," a simple test would look like Example 6–12:

***Example 6–12 A Simple Test***

```
// use the simple form of tests
p1.put("TestForm", Pattern.TEST_FORM_SIMPLE);

//////////////////////////////////////////////////////////////////////
// add a SimpleTest to the Pattern
// emsg.sender == "david@fun.com"
//////////////////////////////////////////////////////////////////////

//create a simple test
SimpleTest simple = p1.getSimpleTestTable().add();

//
// emsg.sender == "david@fun.com"

// set the left side
Expression lhs = simple.getLeft();
lhs.setForm(Expression.FORM_SINGLE_TERM);

lhs.setSingleTermValue("emsg.sender");

// set the operator
simple.setOperator("==");

// set the right hand side
Expression rhs = simple.getRight();
rhs.setForm(Expression.FORM_ADVANCED);

AdvancedExpression radv = rhs.getAdvancedExpression();
radv.insert(0, "\"david@fun.com\"");
```

## 6.4.5 Adding an Action to a Rule

The Rules SDK supports the following types of `Actions`:

- Assert New
- Assert
- Assign
- Call
- Retract
- RL

The setting in the `setForm` property of an action defines the type of action. Add an action using the `add()` method of the `getActionTable` instance associated with a particular rule (see Example 6–13).

***Example 6–13 Defining an Action Type***

```
//////////////////////////////////////////////////////////////////////
```

```
// Add a action to retract the instance of SpamFound
//////////////////////////////////////////////////////////////////////
act = spamRule.getActionTable().add();
act.setForm(Action.FORM_RETRACT);
act.setTarget("spamMessage");  // see above setVariable

//////////////////////////////////////////////////////////////////////
// Add a action to call the kill spam function
//////////////////////////////////////////////////////////////////////
act = spamRule.getActionTable().add();
act.setForm(Action.FORM_CALL);
act.setTarget("kill spam");
//see the datamodel fn definitions, alias for fnKillSpam

Expression exp1 = act.getExpression(0);
Expression exp2 = act.getExpression(1);

exp1.setForm(Expression.FORM_ADVANCED);
exp2.setForm(Expression.FORM_ADVANCED);
AdvancedExpression adv1 = exp1.getAdvancedExpression();
AdvancedExpression adv2 = exp2.getAdvancedExpression();

adv1.setVariable("spamMessage.Spam Email");
adv1.insert(0, adv1.getVariable());

adv2.setVariable("spamMessage.Why is this spam");
adv2.insert(0, adv2.getVariable());
```

### 6.4.6 Notes for Adding RuleSets and Rules

The order in which you add components to a `RuleSet` is important. Use the parent object to create a required child object. For example, after you create a `RuleSet` instance, you can add rules the `RuleTable` instance for the `RuleSet`. After you create the `RuleSet`, then you can add a Rule to the `RuleSet` using the `add()` method for the `RuleTable`. In turn, then add a rule pattern to the rule using the `PatternTable.add()` method. Finally, to create a test in the `Pattern` access the `AdvancedExpression` using `getAdvancedExpression()` or for standard mode tests, use `SimpleTestTable.add()`.

# A

# Oracle Business Rules Files and Limitations

This appendix lists known naming constraints for Rule Author files and names, and certain Rules SDK limitations.

## A.1 Rule Author Naming Conventions

This section covers Rule Author naming conventions.

### A.1.1 Ruleset Naming

Rule Author enforces a limitation for rule set names; a rule set name can only contain the characters (a-zA-Z) and numbers (0-9), or an underscore character (_).

### A.1.2 Dictionary Naming

Rule Author dictionary names can contain both upper and lowercase letters (a-zA-Z), numbers (0-9), periods (.), underscore characters (_), and hyphens (-). Special characters are not valid in a dictionary name.

Rule Author dictionary names are case preserving but case insensitive. This means that the dictionary names "Dictionary" and "DICT" are both valid. This also means that if you create a dictionary named "Test," then you can only create another dictionary named "TEST" if you first delete the dictionary named "Test."

Additionally, dictionary names must contain at least one letter. For example, the dictionary name "1.1" is not valid, but "Version1.1" is valid.

### A.1.3 Version Naming

Rule Author enforces a limitation for the name of a version; a version name can only contain the characters (a-zA-Z), numbers (0-9), or an underscore character (_). Special characters are not valid in a version name.

Rule Author version names are case preserving but case insensitive. This means that the version names `Version` and `VERS` are both valid. This also means that if you create a version named `Test`, then you can only create another version named `TEST` if you first delete the version named `Test`.

### A.1.4 Alias Naming

A Rule Author alias can contain any characters, including a single space. When using an alias in an expression, if the alias begins with a letter, $, or _ and contains only letters, $, _, numbers, and spaces, it does not have to be quoted.

When using an alias containing special characters or embedded spaces in an advanced expression, the alias must be quoted with ` (backquote) characters. For example, the alias `Driver@` must be specified as:

```
`Driver@`
```

### A.1.5 XML Schema Target Package Naming

The **Target Package Name** that you specify for an XMLFact, on the XML Schema Selector page is limited to ASCII characters, digits and the underscore character.

## A.2 Rule Author Session Timeout

You should save the dictionary periodically as you work since Rule Author sessions expire after a period of inactivity specified in the Rule Author application's `web.xml` file using the `<session-timeout>` element.

## A.3 Rules SDK and Rule Author Temporary Files

When working with the Rules SDK or with Rule Author using the file repository, the Rules SDK uses temporary files. Under normal operating conditions, the Rules SDK removes these files from the system when an operation that uses the temporary file completes. It is possible, due to certain abnormal termination conditions for these temporary files to be left on the system.

See Section B.3, "Working with a File Repository" for more information about file repositories and temporary files.

# B

# Using Rule Author and Rules SDK with Repositories

This appendix contains information on using Rule Author and Rules SDK with repositories. The following topics are covered:

- "Working with a WebDAV Repository"
- "WebDAV Repository Security"
- "Working with a File Repository"
- "High Availability for your Repository"

## B.1 Working with a WebDAV Repository

This section contains information on setting up and configuring a WebDAV rules repository.

### B.1.1 Setting up a WebDAV Repository

The Oracle Business Rules SDK supports the use of a WebDAV repository as the persistent storage for rules constructed with the SDK. This appendix briefly mentions some issues for consideration in setting up a WebDAV repository and presents some basic instructions for setting up a file system based WebDAV repository in an Oracle HTTP Server. WebDAV is supported in the Oracle HTTP Server by the `mod_oradav` module. Documentation on configuring and using `mod_oradav` can be found in the *Oracle HTTP Server Administrator's Guide*.

The WebDAV protocol is an extension to the HTTP protocol which enables remote users to write content to the web server. The server should be configured properly to prevent undesirable consequences. For more details, see the section titled "WebDAV Security Considerations" in Chapter 9 of the *Oracle HTTP Server Administrator's Guide*.

It is strongly recommended that some or all of the following be employed:

- Require authentication for access to WebDAV enabled areas.
- Use of SSL, at least during authentication (for the entire session if Basic Authentication is used).
- Use of the `ForceType` directive to prevent execution for URLs that reference content in WebDAV enabled areas.

The following example demonstrates the steps required to establish a WebDAV based rules repository in Oracle HTTP Server where the content is stored in the file system. All file system paths in this example are relative to the *`ORACLE_HOME`* in which the Oracle HTTP Server is installed. This example also assumes that the user is logged in

as the user who installed Oracle Application Server, and that Oracle HTTP Server can be accessed with the URL `http://www.myserver.com:7777`.

> **Note:** This example configuration for the WebDAV repository should only be used for internal testing and not for an actual production environment. This configuration does not configure access control and therefore allows anyone to access and modify the WebDAV repository. Please refer to Section B.2 for information about configuring WebDAV repository security.

1. Navigate to the `Apache/Apache/htdocs` directory (folder).
2. Create a directory named `rule_repository`.
3. Ensure that Oracle HTTP Server can read and write to the `rule_repository` directory.
4. Navigate to the `Apache/oradav/conf` directory.
5. Edit the `moddav.conf` file and add the following lines:

```
<Location /rule_repository>
    DAV on
    ForceType text/plain
</Location>
```

6. Restart Oracle HTTP Server (see the section titled "Starting, Stopping, and Restarting Oracle HTTP Server" in Chapter 1 of the *Oracle HTTP Server Administrator's Guide*.

These instructions establish a WebDAV repository accessible with the following URL:

`http://www.fully_qualified_host_name.com:7777/rule_repository/`

> **Note:** In order for authentication to work, you must use a fully qualified host name in the URL.

### B.1.2 Connecting to a WebDAV Repository

Selecting WebDAV as the repository type in Rule Author presents the configuration parameters shown in Table B–1:

***Table B–1 Configuration Parameters for Connecting to a WebDAV Repository***

| Parameter | Description |
|---|---|
| URL | The URL for the desired WebDAV rule repository. This is a required parameter. The host name must be a fully qualified host name. |
| Proxy Host | The host name of the proxy server. This is required only if a proxy server is present between Rule Author and the WebDAV server. |
| Proxy Port | The port number to use on the proxy server. This is required only if a proxy server is present between Rule Author and the WebDAV server. |

## B.2 WebDAV Repository Security

WebDAV allows read and write access to a WebDAV enabled server. It is highly recommended that steps are taken to secure the WebDAV server. To this end, it is likely

that connections to a WebDAV server will need to be encrypted using SSL, thus requiring authentication in order to establish the connection.

### B.2.1 Communicating with a WebDAV Repository Over SSL from Rule Author

Basic SSL connections to a WebDAV repository are supported in Rule Author when Rule Author has been deployed in an Oracle Application Server environment. All that is required is that the WebDAV URL entered specify `https`.

If Rule Author is deployed in a standalone OC4J environment, or is deployed in a non-Oracle container that supports only HTTP, then SSL connections to a WebDAV repository are not supported.

Oracle Application Server comes with a test SSL certificate that is self-signed. This certificate should be replaced with your own certificate because it is not secure to use this test certificate in a production environment. If you use a certificate from a trusted authority, WebDAV access is available from both within and outside of the OC4J container. If you choose to use a self-signed certificate of your own, access from within the container is available but from outside the container, your default JSSE trust store must be modified in order to gain access. Refer to the *JSSE Reference Guide* included in the JDK for details.

Additionally, the Oracle SSL implementation must not be present in the classpath of the J2SE application.

### B.2.2 Setting the Location of your Oracle Wallet

To customize the location of your Oracle wallet for Rule Author:

1. Login to Enterprise Manager and go to the OC4J home page.
2. Click the **Applications** tab.
3. Click the link to your Rule Author application (the name of this link was defined when you first deployed the Rule Author application).
4. Click the **ruleauthor** link in the "Modules" table.
5. Click the **Administration** tab.
6. In the "Mappings" task, find row labeled "Environment Entry Mappings," then click the corresponding icon in the "Go to Task" column.
7. Specify your desired wallet location in the "Deployed Value" column for `walletStorePath` entry.
8. Restart Rule Author.

You can also set your wallet location at the time you deploy Rule Author by clicking on "Edit Deployment Plan" and then expanding the navigation tree on the left until "env-entry" is visible. Expand "env-entry" and then select `walletStorePath`. Be sure to restart Rule Author after you specify your desired wallet location.

### B.2.3 Configuring Rule Author for WebDAV Repository Authentication

When Rule Author attempts to connect to a WebDAV repository that has been configured to require authentication, Rule Author must be able to respond to the authentication request. Configuring Rule Author for repository authentication consists of the following steps:

1. Store the appropriate WebDAV repository user name and password in an Oracle Wallet.

2. If a proxy server is present and it also requires authentication, store the proxy server user name and password in the Oracle Wallet.
3. Configure the Rule Author environment entry to point to the Oracle Wallet (see Section B.2.2, "Setting the Location of your Oracle Wallet").
4. Restart the Rule Author application.

### B.2.4 Storing Data in an Oracle Wallet for WebDAV Repository Authentication

When a request for authentication from a WebDAV repository is received, the following information is provided:

- The host name of the server requesting authentication.
- The port on the server.
- The realm (or `AuthName` in Oracle HTTP Server configuration).
- An indication of whether or not this is proxy server authentication.

This information is used to construct keys for retrieving the user name and password for authentication. If there is a proxy server present and it requires authentication, multiple authentication requests may be processed: one for the proxy server and one for the WebDAV server.

If the request is for proxy authentication, the keys begins with "proxy-". This is followed by the host name, port, and realm (in that order) with a "-" separating each field. Finally, "-u" is appended to the key for the user name and "-p" is appended for the password. For example, given the following:

- Host is `myserver.myco.com`
- Port 443
- Realm is "Authorized WebDAV Users Only"
- A proxy server is present: `wwwproxy.myco.com`
- Proxy port is 80
- Proxy realm is "Authorized Proxy Users Only"

The keys for proxy authentication would be:

- For the user: "proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-u"
- For the password: "proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-p"

The keys for WebDAV authentication would be:

- For the user: "myserver.myco.com-443-Authorized WebDAV Users Only-u"
- For the password: "myserver.myco.com-443-Authorized WebDAV Users Only-p"

The user name and password are entered into an Oracle wallet with the `mkstore` command which is in the `bin` directory of the *`$ORACLE_HOME`*. Creating and modifying the Oracle wallet requires a password which is specified when the wallet is created. However, the wallet is constructed such that a password is not required at runtime to lookup the user name and password. Therefore, in order to protect this sensitive data, file system permissions must be used to restrict access. Access should be granted to only the user that must access the wallet at run time. The `mkstore` command creates the wallet with restricted permissions by default.

The following commands create a wallet in a the `/wallets` directory and store the user names and passwords for the example shown above where the user names and passwords are `proxyUser`, `proxyPassword`, `webdavUser`, and `webdavPassword`:

```
mkstore -wrl /wallets/rules_wallet -create
mkstore -wrl /wallets/rules_wallet -createEntry
'proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-u' proxyUser
mkstore -wrl /wallets/rules_wallet -createEntry
'proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-p' proxyPassword
mkstore -wrl /wallets/rules_wallet -createEntry 'www.myco.com-80-Authorized WebDAV
Users Only-u' webdavUser
mkstore -wrl /wallets/rules_wallet -createEntry 'www.myco.com-80-Authorized WebDAV
Users Only-p' webdavPassword
```

Each command prompts you for the wallet password and, if needed, creates the directory for the wallet (`rules_wallet` is a directory).

The following command prints a usage message listing various capabilities of the `mkstore` command:

```
mkstore -help
```

# B.3 Working with a File Repository

This section contains information about setting up and working with file repositories.

## B.3.1 Setting up a File Repository

Oracle Business Rules supplies a blank file repository that does not contain a dictionary. This file repository is named `emptyFileRepository` and is located in the *`$ORACLE_HOME`*`/rules/lib` directory.

To setup a new file repository, copy and rename the `emptyFileRepository` file. Then, provide this file name and location in the Repository Connect page (see Section 2.4.1, "Connecting to a Rule Author Repository").

After you create a new file repository, you can connect to the new file repository and then create and save dictionaries in the repository.

## B.3.2 File Repository Updates and Temporary Files

When the SDK invokes the `RepositoryConnection` interface to update repository content, the following occurs:

1. A temporary file is created that contains the updated content. This temporary file is required as the process of rewriting the JAR file may involve reading unread entries from the current repository. It also provides a measure of safety should something go wrong writing the new content. The temporary file is created using the `File.createTempFile` method. If the name of the repository is less than three characters long, "_tmp_" is appended. The `File.createTempFile` method requires that the name be at least three characters long. The Sun JDK appends a number to the name; the behavior of other JVMs may differ. The file name extension is ".tmp" and the file is created in the same directory as the existing repository. In summary, the temporary file name of a repository called `myRepository` would be `myRepository65146.tmp`, and the temporary file name of a repository called `rr` would be `rr_tmp_65147.tmp`.
2. The content is written to the temporary file.

3. The existing repository is renamed as the name of the existing repository appended with "_o_r_i_g_" and the current time (UTC) in milliseconds.
4. The temporary file is renamed as the name of repository (for example, `myRepository`).
5. The renamed repository (containing the previous content) is removed.

If an error occurs in this process, cleanup is attempted. If the temporary file was created and still exists, an attempt is made to delete it. If the existing repository was renamed, an attempt is made to restore its original name.

In the event that the temporary file is left behind, the file repository prior to the update attempt should still exist. The temporary file should be deleted as the state of its contents is unknown.

In the event that the renamed repository file is left and the repository file is no longer exists, the renamed repository file contains the content prior to the update and a manual step is required to restore it (namely, renaming or copying the renamed file back to the correct name).

## B.4 High Availability for your Repository

After configuring your WebDAV or file repository, you should add the repository to the OracleAS Recover Manager configuration so that the repository is included in the backup and recovery process.

For more information about this tool, see *Oracle Application Server Administrator's Guide*.

# C

# Oracle Business Rules Frequently Asked Questions

This appendix contains frequently asked questions about Oracle Business Rules. Answers to each question are provided in each of the following categories:

- Frequently Asked Questions About Rules Operations
- What JAR Files are Required for Working with Oracle Business Rules?

## C.1 Frequently Asked Questions About Rules Operations

This section addresses frequently asked questions relating to the semantics of Rules operations in Oracle Business Rules.

### C.1.1 Why is the State of a Fact in a Rule Action Inconsistent with the Rule Condition?

The object was modified between the time the rule was activated and the time the rule was fired (executed), and the object was not re-asserted in the Rules Engine.

Objects (Java or RL) must be asserted as facts in the Rules Engine before they are used in rule evaluations. When an object that has been asserted as a fact is modified, either in the action of a rule or by something external to the Rules Engine (presumably by the application), the object must be re-asserted in the Rules Engine in order for the current object state to be reflected in the Rules Engine and thus in the rule evaluation. If this is not done, the application and Rules Engine are in an inconsistent state which can lead to unexpected behavior.

A Java bean may be written to support `PropertyChangeListener` so that the Rules Engine can automatically maintain a consistent state when a bean property us update. For more information, see Section 1.3.4.1, "Java Fact Type Definitions".

The one exception to this rule is for an object whose content is not being evaluated; that is, the Rules Engine does not contain a rule that tests or accesses any method or property of that object. One example of such a case is an object used to accumulate results from rule evaluations.

> **Tip:** Suppose you have a rule that produces a sum from a collection of facts. Re-asserting the facts whose values are being summed yields an incorrect result in the fact containing the sum. Make sure you also re-assert the rule that produces the sum.

### C.1.2 A Changed Java Object was Asserted as a Fact, but no Rules Fired. Why?

The object must be re-asserted in the Rules Engine. Therefore, the Rules Engine did not re-evaluate any rule conditions and did not activate any rules. For more information, refer to Section C.1.1.

### C.1.3 What are the Differences Between Oracle Business Rules RL Language and Java?

See Appendix A in *Oracle Business Rules Language Reference Guide*.

## C.2 What JAR Files are Required for Working with Oracle Business Rules?

Oracle Business Rules support requires the JAR files listed in Table C–1. All paths are relative to *$ORACLE_HOME*.

***Table C–1 Oracle Business Rules Required JAR Files***

| JAR File | Description |
|---|---|
| `rules/lib/rl.jar` | The Oracle Business Rules Rules Engine library. This is the Java API used to instantiate and interact with the Rules Engine. |
| `rules/lib/rl_dms.jar` | Rules Engine Dynamic Monitoring Service (DMS) support. This file is required if DMS is enabled for a `RuleSession`. |
| `rules/lib/rulesdk.jar` | The Oracle Business Rules SDK. This is the Java API used to programatically author rules. |
| `rules/lib/webdavrc.jar` | The Oracle Business Rules SDK library for support of WebDAV repositories. This file is required when using the SDK with a WebDAV repository. |
| `rules/lib/jr_dav.jar` | The WebDAV client library. This file is required when using the SDK with a WebDAV repository. |
| `jlib/oraclepki.jar` | This file is required to support authentication with a repository such as a WebDAV repository. |
| `jlib/ojpse.jar` | This file is required to support authentication with a repository such as a WebDAV repository. |
| `rules/lib/jsr94.jar` | The standard JSR-94 library. |
| `rules/lib/jsr94_obr.jar` | The Oracle Business Rules JSR-94 implementation. |
| `LIB/xml.jar` | This file is required by the Rules SDK. |
| `LIB/xmlparserv2.jar` | This file is required by the Rules SDK. |
| `j2ee/home/lib/http_client.jar` | This file is required when using the Rules SDK with a WebDAV repository. |

# D

# Oracle Business Rules Troubleshooting

This appendix contains workarounds and solutions for issues you may encounter when using Oracle Business Rules. The following topics are covered:


- Public Fact Variables are not Accessible with Rule Author
- Global Variables may not be Used in RL Functions
- Importing JDK 1.4.2 Classes
- Managing Popup Windows on Firefox
- Using the String Data Type with Methods
- Preserving Class Order and Hierarchies in the Data Model
- Validating Generated RL from Rule Author
- Using RL Reserved Words as Part of a Java Package Name
- Getter and Setter Methods are not Visible
- XML Facts not Asserted at Runtime


## D.1 Public Fact Variables are not Accessible with Rule Author

Public fact variables are not accessible with Rule Author. For example, the variables in the following class would be accessible with Oracle Business Rules RL Language but not with Rule Author:

```
public class Test {
   public int i = 0;
   public String s = "string";
}
```

No variable can be accessed in the Rule Author for facts of type Test. In order to access these variables, methods like the following need to be added:

```
public void setI(int i) { this.i = i; }
public int getI() { return i; }
public setB(boolean b) { this.b = b; }
public boolean isB() { return b; }
```

Note that no variable `i` is required for `setI(int i)` and `getI()` to work properly. For more information, please refer to the Sun Microsystems Java Bean specification.

## D.2 Global Variables may not be Used in RL Functions

For RL generated from the SDK (for example, Rule Author), global variables may not be referred to directly in an RL function.

To work around this issue, if an RL function needs to access a global variable, the global variable should be passed as a parameter to the RL function. The parameter name allows access to the global variables inside the RL function body.

## D.3 Importing JDK 1.4.2 Classes

If you choose to run Rule Author using JDK 1.4.2, be aware that Java classes compiled using JDK 1.5 do not import properly. If you try to import Java classes compiled using JDK 1.5 into Rule Author using JDK 1.4.2, an error message like the following appears:

```
Cannot perform operation. 'RUL-01527: Received exception for loadClass.
RUL-01016: Cannot load Java class example7.Example7. Please make sure
the class and all its dependent classes are either in the class path,
or user specified path. Root Cause: example7/Example7 (Unsupported
major.minor version 49.0) '
```

To work around this issue, run Rule Author using JDK 1.5 or recompile the classes using JDK 1.4.2.

## D.4 Managing Popup Windows on Firefox

If you are running Rule Author on Firefox browser, you may encounter a problem if you close many popup windows using the X button in the upper corner of the window instead of the **OK**, **Cancel**, or **Apply** buttons.

The easiest way to avoid this problem is to use the **OK**, **Cancel**, or **Apply** buttons instead of the window controls to close the popup windows. You can also change value of the `dom.popup_maximum` parameter to allow for many more popup windows. To do this:

1. Type `about:config` as the URL and locate the `dom.popup_maximum` parameter.
2. Set the value to 10000 or higher.

## D.5 Using the String Data Type with Methods

The built-in data type `String` does not contain any methods. Thus, if `x` is a String, `x.substring(1)` would be invalid in an advanced expression.

To work around this issue:

1. Import `java.lang.String` into the data model as a Java fact type.
2. Give this fact type an alias. The default alias is `java_lang_String`.
3. Use this new fact type instead of `String` when you are defining RL fact types or variables in the data model.

## D.6 Preserving Class Order and Hierarchies in the Data Model

Classes and interfaces used in Rule Author must follow the following rules:

1. If you are using a class or interface and its superclass, the superclass must be declared first. Otherwise, the generated RL program throws an exception like the following:

```
"FactClassException: fact class for 'pkg.Parent' should be declared earlier
in rule session".
```

2. If you are using a class or interface, only its superclass or one of its implemented interfaces may be mentioned. If multiple interfaces are mentioned, the generated RL Language program throws an exception like the following:

```
MultipleInheritanceException: fact class 'pkg.Child' cannot extend both
'pkg.ParentInterface' and 'pkg.ParentClass'
```

To work around these issues:

1. Identify the hierarchy of classes and interfaces in the data model you want to use in your rule sets.
2. For each class or interface in the hierarchy, check the **Support Xpath Assertion** box. This causes fact class statements to be generated in the correct order as part of the data model RL.

## D.7 Validating Generated RL from Rule Author

In order to validate generated RL from Rule Author, make sure that the Java classes in the Data Model are in the OC4J classpath. For more information on setting the OC4J classpath, see Section 3.10, "Working with Test Rulesets".

## D.8 Using RL Reserved Words as Part of a Java Package Name

Invalid RL Language is generated if an RL Language reserved word (for example, the word `rule` in `mypkg.rule.com`) is part of the Java package name. If an RL Language reserved word is used in a Java package name, an error message like the following appears:

```
Oracle RL 1.0: syntax error PareseException: encountered 'rule' when expecting
one of: <XML_IDENTIFIER> ...<IDENTIFIER> ... "*" at line 11 column 19 in main
```

There is no workaround for this issue; do not use RL Language reserved words in Java package names.

## D.9 Getter and Setter Methods are not Visible

Rule Author does not list the methods supporting a Java bean property in choice lists; only the bean properties are visible. For example, a Java bean with a property named "Y" must have at least a getter method (`getY()`) and may also have a setter method (`setY(y-type-parm)`). All of properties and methods (including getter and setter that compose the properties) are displayed when viewing the Java FactType. Only the properties of Java Classes (not the getter and setter methods) are displayed in choice lists. When attempting to control the visibility of the property it is best to use the properties visibility flag. Marking a getter or a setter method as not visible may not remove the properties from choice lists.

There is no current workaround for this issue.

## D.10 XML Facts not Asserted at Runtime

The XML Fact page for an XML Schema generated class shows the **Support XPath Assertion** box. This box is checked by default. Un-checking this box and saving your changes marks the XML Fact as not supporting XML style assertion, which in turn

means that any instance of this type and any of its children are not asserted by a call to assertXPath for an XML document.

There is no workaround for this issue; you should make sure the **Support XPath Assertion** box is checked for all XML FactTypes.

# Index


## A

accessing Fact variables, D-1
advanced test expression
  rule author
    advanced test expression, 3-11
Advanced Test Expression option, 3-10
alias
  naming, A-1
asserting
  XPath, 4-26
asserting facts with SDK, 2-28
assertXPath function, 4-26

## B

business rules
  definition, 1-2
business vocabulary
  defining in data model, 2-13

## C

classpath
  adding, 2-9
connecting
  to a repository, 2-5, 4-2
constraint
  definition, 1-8, 3-2
  enumeration, 3-2
  range, 3-2
  regular expression, 3-2
creating
  repository, B-5
customization
  rule, 2-19, 2-24, 3-2

## D

data model
  defining XML, 4-6
  definition, 1-2, 2-9
  generating, 3-10
  saving, 3-10
definitions
  constraint, 3-2
  JavaFact, 3-8
  RL function, 3-6
  RLFact, 3-5
  variable, 3-1
  XMLFact, 4-6, 4-11
dictionary
  creating, 2-4, 4-2
  deleting, 3-12
  description of, 1-7
  Dictionary Directory field, 2-7, 4-3
  Dictionary Name field, 2-7, 4-3
  exporting, 3-13
  importing, 3-13
  loading, 2-26, 2-27, 6-3
  naming, A-1
  saving, 2-8, 4-4, 4-5

## E

EmptyFileRepository, B-5

## F

fact type
  Java, 2-9
  RLFact, 3-5
  XML, 1-7
file repository
  creating, 2-5, 4-2
  establishing access to, 6-3
  initialization parameters, 6-2
  repository type key, 6-2
  temporary files, B-5
  updating content, B-5
forward chaining, 1-3
forward chaining system, 1-3
frequently asked questions, C-1

## H

high availability for your repository, B-6

## I

importing, 2-26
  Java classes, 2-11
importing JDK 1.4.2 classes, D-2

inference cycle, 1-3
initialization parameters for a file repository, 6-2
initialization parameters for a WebDAV repository, 6-2

## J

Jar file
  repository, 1-4
Java classes
  importing into data model, 2-11
Java fact type, 1-7, 2-9
JAXB generated classes, 4-6
JDK 1.4.2 classes
  importing, D-2
JSR-94
  extensions, 5-8
  rule execution set, 5-1
  with RL Language text, 5-3
  with Rule Author rules, 5-1
  with URL, 5-5

## L

loading a dictionary, 6-3
Logging option, 3-10

## M

method object chaining, 3-10
`mod_oradav` module, B-1

## N

naming conventions
  alias, A-1
  dictionary, A-1
  ruleset, A-1
  version, A-1

## O

object chaining
  expand box, 3-10
object visibility, 3-9
options
  Advanced Test Expression, 3-10
  Logging, 3-10
  Use Alias, 3-10
Oracle Business Rules
  required JAR files, C-2
Oracle Business Rules RL language, 1-5
Oracle wallet
  setting the wallet location, B-3

## P

property object chaining, 3-10
property visibility
  object, 3-9

## R

remove
  ruleset, 2-16
repository
  backup and recovery, B-6
  connecting, 2-5, 4-2
  description of, 1-7
  emptyFileRepository, B-5
  Jar file, 1-4
  WebDAV, 1-4
repository type key
  for file repository, 6-2
  for WebDAV repository, 6-2
required JAR files for Oracle Business Rules, C-2
results
  using container objects, 3-19
  using global variables, 3-18
  using reasoned on objects, 3-20
Rete algorithm, 1-3
RL language
  generating, 3-10
  saving, 3-10
RL tab, 3-10
Rule Author
  Home page, 2-4
  how to start, 2-2
  introduction, 1-4
  Login page, 2-2
  rules, 1-6
  session-timeout, A-2
  starting, 2-2
  temporary files and file repositories, A-2
  working with Test Rulesets, 3-14
rule firing, 1-3
rule session
  asserting facts, 2-28
  executing, 2-28
  using run function, 2-29
rule-enable
  Java application, 1-9, 2-25
rules
  adding a pattern to, 2-18
  adding actions, 2-21
  customization, 2-19, 2-24, 3-2
  data driven, 1-3
  defining, 2-16
  defining tests for patterns, 2-19
  engine, 1-5
  forward chaining, 1-3
  name field, 2-16
  priority field, 2-16
  rule actions, 1-6
  rule conditions, 1-6
rules SDK, 2-26, 6-1
  introduction, 1-5
ruleset
  defining, 2-15
  naming, A-1
  removing, 2-16
run function, 2-29

## S


SDK
  classes, 6-1
  executing a rule session, 2-28
  generating RL, 2-27
  introduction, 1-5, 6-1
  pattern, 6-8
  rules, 6-6
  rulesets, 6-6
  working with data model, 6-3
session-timeout, A-2
setting your Oracle wallet location, B-3
starting Rule Author, 2-2


## T


Test Rulesets feature, 3-14
troubleshooting, D-1


## U


Use Alias option, 3-10


## V


version
  naming, A-1
visible field
  method visibility, 3-9


## W


WebDAV repository, 1-4
  establishing access to, 6-2
  how to connect, B-2
  how to set up, B-1
  how to set up security, B-2
  initialization parameters, 6-2
  repository type key, 6-2
web.xml
  session-timeout, A-2


## X


XML document
  unmarshalling, 4-24
XML fact types, 1-7
XMLFact
  adding, 4-6
  asserting, 4-26
  importing schema, 4-9
  importing with SDK, 4-24
  JAXB class directory, 4-7
  JAXB generated classes, 4-6
  JAXB unmarshalling, 4-24
  target package name field, 4-7
  XML Schema field, 4-7