Oracle® Business Rules

User's Guide

10*g* (10.1.3.1.0)

B28965-02

September 2006

ORACLE®

Oracle Business Rules User's Guide, 10*g* (10.1.3.1.0)

B28965-02

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

Contents

3 Working with Rule Author Features

4 Using XML Facts with Rule Author

5 Using JSR-94

C Oracle Business Rules Frequently Asked Questions

D Oracle Business Rules Troubleshooting

Index

Preface

This Preface contains these topics:

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://www.oracle.com/technology/membership/index.html`

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://www.oracle.com/technology/documentation/index.html`

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 about 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. By using Oracle Business Rules, business analysts can change business rules 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:

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 the age of a driver is younger than 21, then decline to rent.

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

If a frequent flyer account has total miles for the year that 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, you can use Oracle Business Rules to 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 nontechnical 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 must modify the application, recompile, and then redeploy the application. Using Oracle Business Rules, this process can be simplified because a business rules application is built to support easily changing business rules.

Oracle Business Rules allows a business analyst to change policies that are expressed as business rules, with little or no assistance from a programmer. Applications using Oracle Business Rules, called **rule-enabled applications**, support continuous change that allows the applications to 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, constrain and support facts. In Oracle Business Rules, a **data model** specifies the types of facts or business objects that you can use to create business rules. For example, for a car rental company that needs to create a rule to match the age of a driver, the driver information represents the facts used in the rule. Using Rule Author, you can define 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 in a single network of nodes in current working memory.

By using the Rete algorithm, the Rules Engine avoids unnecessary rechecking when facts are deleted, added, or modified. To process facts and rules, the Rete algorithm creates and uses an input node for each fact definition and 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 affecting 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. With Oracle Business Rules you add a set of facts to the system using assert calls.

[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; this requires only an incremental amount of work.

- 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.

In 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. These new facts 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 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**.

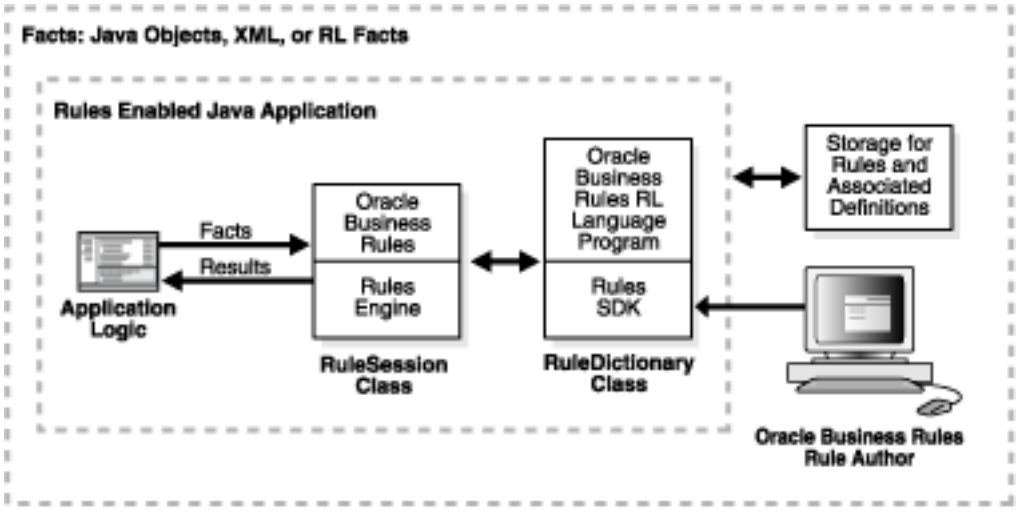
1.2 Oracle Business Rules Components

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. It provides a point-and-click interface for creating new rules and editing existing rules. Rule Author allows 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 you 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 either to work

with the initial set of rules, or to 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.

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

Note 1: It is not safe for multiple users to edit the same dictionary.

Note 2: 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 Rules SDK to create, modify, and access rules and the data model using well-defined interfaces. Customer applications can use 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 rules, 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 (Rule Author uses the Rules SDK dictionary storage plug-in to store rules).

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

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). 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. 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 about RL Language

1.2.4 Introducing Oracle Business Rules Rules Engine

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

Rules Engine has the following features:

- High performance: Rules Engine implements specialized matching algorithms for facts that are defined in the system.
- Thread-safe execution suitable for a parallel processing architecture: Rules Engine provides one thread that can assert facts while another is evaluating the network.
- Agility: Rules Engine allows rules to change without stopping business processes.

A rule-enabled Java application can load and run rules programs. The rule-enabled application passes facts and rules to the Rules Engine (facts are asserted in the form of Java objects or XML documents). The Rules Engine runs in the rule-enabled Java application and uses the Rete algorithm to efficiently fire rules that match the facts.

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

1.3 Oracle Business Rules Rule Author Terms and Concepts

This section provides information about 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 the age of a driver 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 a combination of facts 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. A rule might perform several kinds of actions. 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 1-7

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

In Oracle Business Rules, a repository stores dictionaries. A dictionary usually corresponds to a rules application and typically stores the rules and definitions for a rule-enabled application. A dictionary is a set of XML files that stores the rules 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 Rules SDK. Dictionaries may 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

In Oracle Business Rules, facts that you can run against the rules are data objects that have been asserted. 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.

In Rule Author, you make business objects and their methods known to Oracle Business Rules using fact definitions that are part of a data model.

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. 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 Rules Engine so that rules can access, create, modify, and delete instances of the Java class. Declaring a Java fact type allows 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 Rules Engine can detect that a Java object supports `PropertyChangeListener`; in this case it uses 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 subelements of an XML element or `complexType` to be declared to 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 4-2

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 JavaBean 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 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 to specify 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 rules could specify other ways that a customer becomes a Gold Customer.

1.3.5 Working with Functions Variables and Constraints

This section covers the following definitions:

- Function Definitions
- Variable Definitions
- Constraint Definitions

1.3.5.1 Function Definitions

In Oracle Business Rules you define a function in a manner similar to a Java method, but a RL function does not belong to a class. You can use RL 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 an RL function definition to share the same or a similar expression among several rules, and to return results to the application.

1.3.5.2 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.

1.3.5.3 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. In 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).

1.4 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 Oracle 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 must determine the parts of rules that are likely to require 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 and the programmer together working determine the business facts required for use with the business rules. These could be the business objects that represent policies that require frequent change, or other policies that might change due to agile business processes.

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 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 or another tool to define 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 business vocabulary.

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 Rules SDK to create or modify rules, or the data model from within the administrative portion of a rule-enabled application.

1.4.5 Modify or Create Application Logic That Uses Oracle 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 Rules Engine. Otherwise, the programmer may need to invoke 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 Rules Engine.

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

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 application. The programmer must provide a set of tests and must in the debugging for a complex set of rules. The programmer can enable Rules Engine tracing to provide information about facts, rule activations, and rule firings. The programmer should develop an automated mechanism for loading test facts 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 and 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 includes the following sections:

Note: We call the sample application that we build in this chapter the Java How-To. This sample application is available in the Oracle Business Rules area, under the Viewlets and Tutorials heading, on the Oracle Technology Web site,

`http://www.oracle.com/technology/products/ias/business_rules/files/how-to-rules-java.zip`

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 about 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 "Users" column to add a user.
7. 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 for Rule Author authentication to work, the Rule Author user must belong to the `rule-administrators` role.

8. Restart the Rule Author application.

2.2 Starting Rule Author

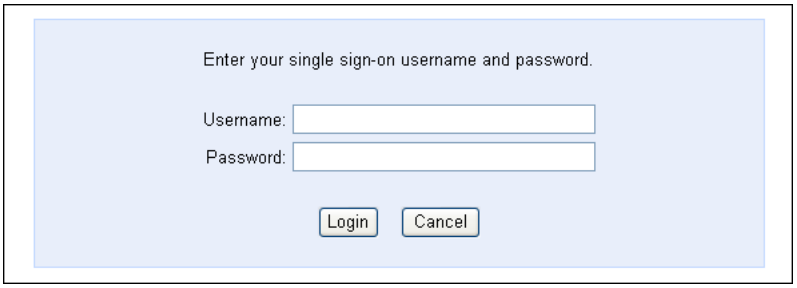
Start Rule Author by entering the URL for the home page. The URL for the home page typically includes the name of the host computer and the port number assigned to the application server during the installation, plus the path of the Rule Author home page. For example:

```
http://myhost1.mycompany.com:8888/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.

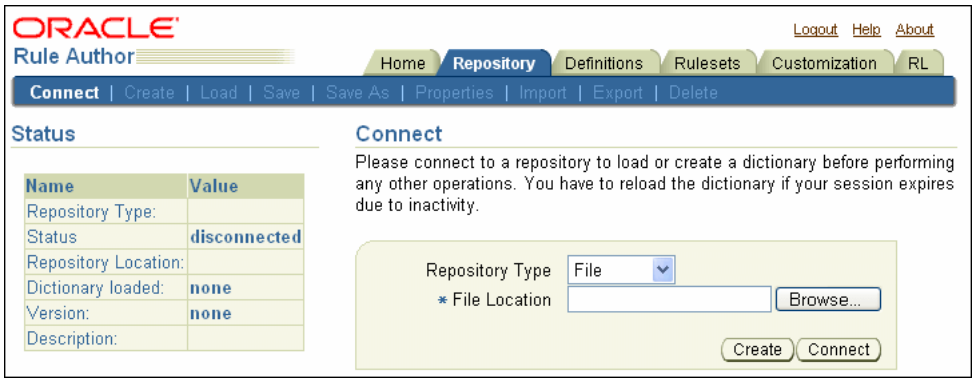
Figure 2–1 shows the Rule Author login page. Specify the user name and password you supplied when you created the Rule Author user (Section 2.1, "Creating a Rule Author User").

Figure 2–1 Rule Author Single Sign-on Login Page



After logging in, click the Repository tab to 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.

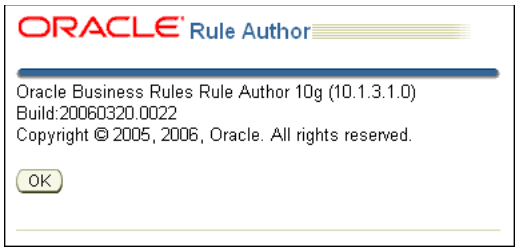
Figure 2–2 Initial Rule Author Repository Connect Page



Each Rule Author page provides the Logout, Help and About links. You can use these links as follows:

- 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 (see Figure 2–1).
- Click **Help** to access online help for the Rule Author.
- Click **About** to view Rule Author version and build information (Figure 2–3). Click **OK** to dismiss this window.

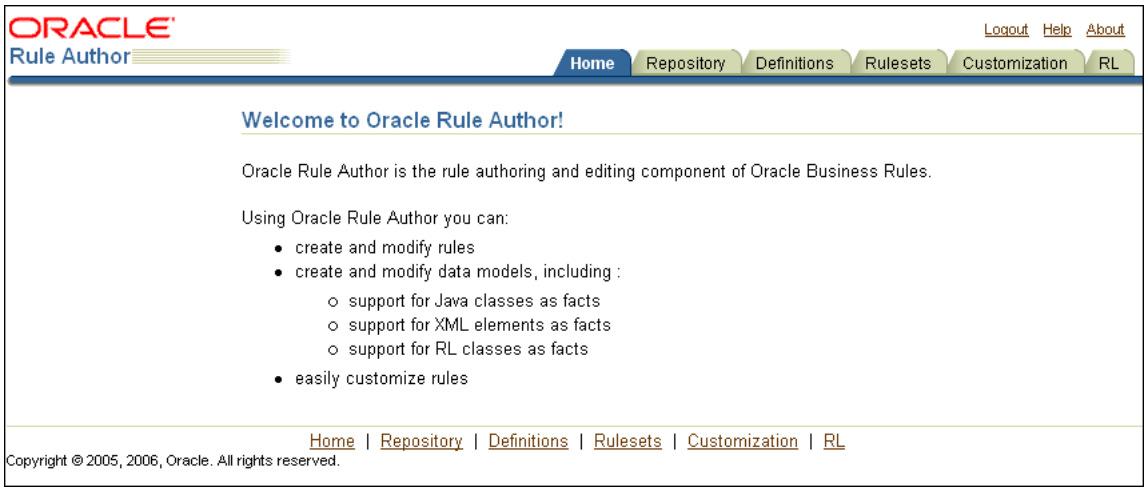
Figure 2–3 About Rule Author



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 must 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 Java 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 Java 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

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

Note: If you choose to use a WebDAV repository, the repository must exist before you can connect to it. See Appendix B, "Using Rule Author and Rules SDK with Repositories" for more information.

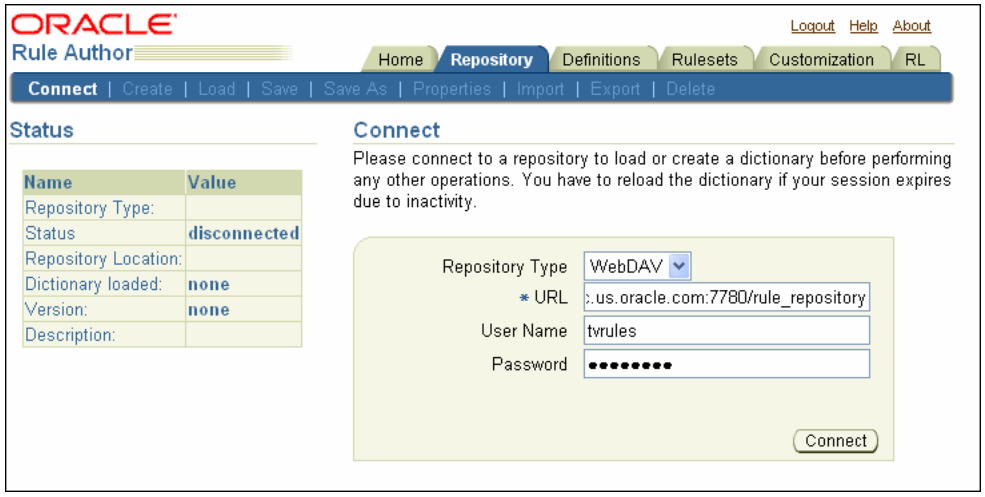
To connect to a repository, do the following:

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:`*`port/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

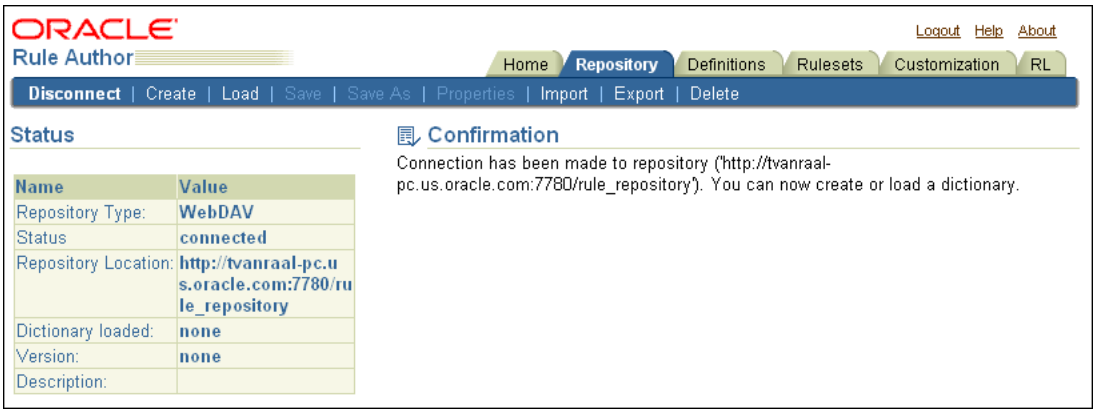


See Section B.1, "Working with a WebDAV Repository" for information on how to setup a WebDAV repository.

5. If there is a proxy between the system where Rule Author runs and the WebDAV server, then Rule Author also shows the **Proxy User** and **Proxy Password** fields.
6. Click **Connect**. After you click connect, Rule Author shows a confirmation message (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



See Also: "Connecting to a WebDav Repository Using a Proxy" on page B-3

2.4.2 Creating a Rule Author Dictionary

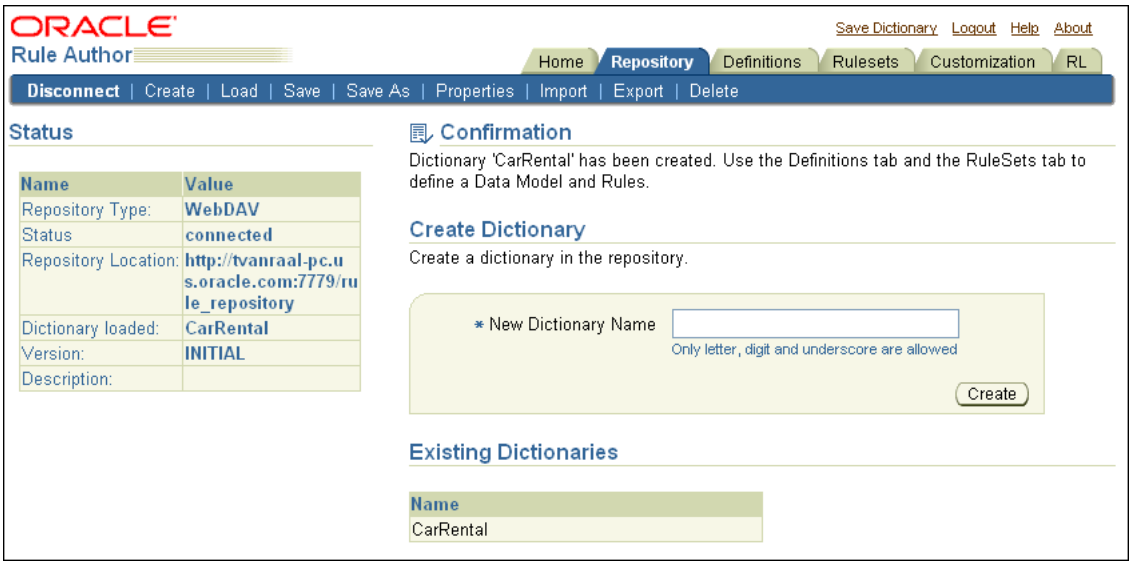
A Rule Author dictionary is the top-level container and the starting point for working with Rule Author. A dictionary usually corresponds to the rules portion of an application.

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

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 confirmation 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 Java How-To. This dictionary is located in the `CarRepository` file repository in the `/dict` directory. To use this dictionary, do the following:

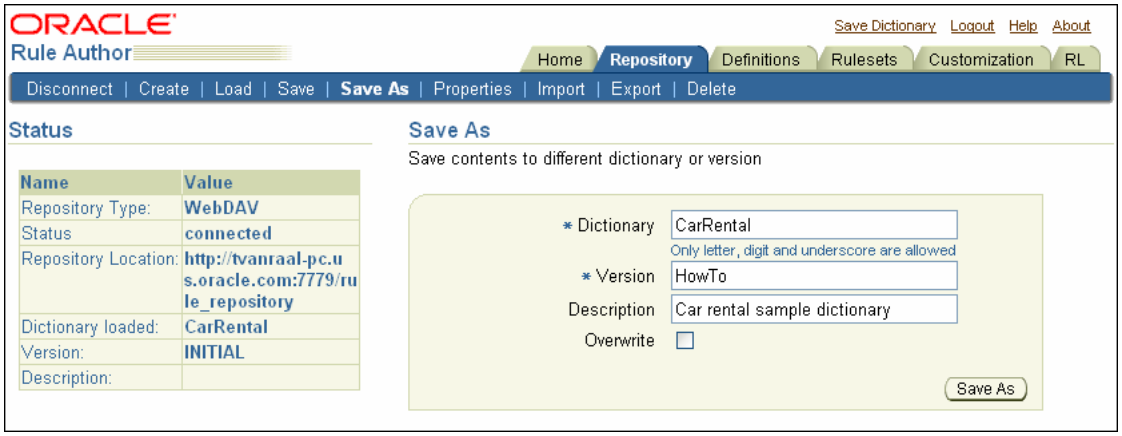
1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. Select **File** in the **Repository Type** box.
4. Enter the complete path to the file repository in the **File Location** field. For example, enter `C:/demo/dict/CarRepository`.
5. Click **Connect**.

2.4.3 Saving a Rule Author Dictionary with a Version

To save to a different dictionary name or to 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, enter `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**, Rule Author shows a confirmation message.

Figure 2–8 Rule Author Save As Page



Note: Rule Author allows you to use **Save As** to overwrite a dictionary with the same name and version. Select the **Overwrite** check box to save a dictionary with the same name and version.

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 because 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 and for details on how Rule Author automatically saves the current work to a dictionary version when a timeout occurs

2.5 Defining a Data Model for the Car Rental Sample

Before working with rules, you must 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 4-10

2.5.1 Using Java Objects as Facts in the Car Rental Sample

The Java 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`, where $*HowToDir* is the directory where you installed the Java How-To.

2.5.2 Adding Java Classes and Packages to Rule Author

Before you can import Java facts into a data model you must 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:

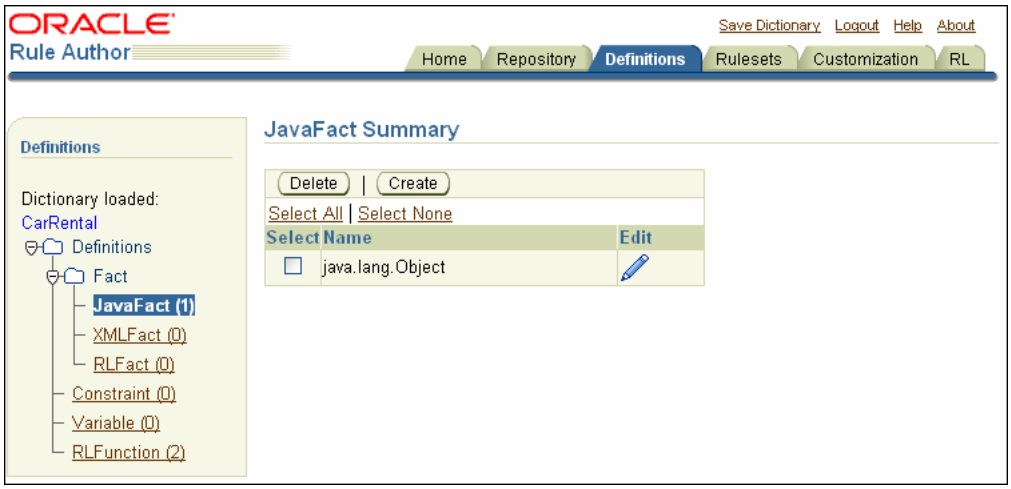
1. Click the **Repository** tab.
2. Skip to Step 3 if you just created the dictionary as shown in Section 2.4.3. On the Load page, select the `CarRental` dictionary and `HowTo` version, then and click **Load**.
3. Click the **Definitions** tab. The navigation tree shows the Definitions folder that contains the available definitions. Nothing is shown in the main pane.

   Note: In Rule Author, the bottom pane usually contains a navigation tree and a content area (the main pane). When you select the **Definitions** tab, the Definitions folder appears at the top of the tree.

4. The Definitions folder in the tree contains the Fact folder, which 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

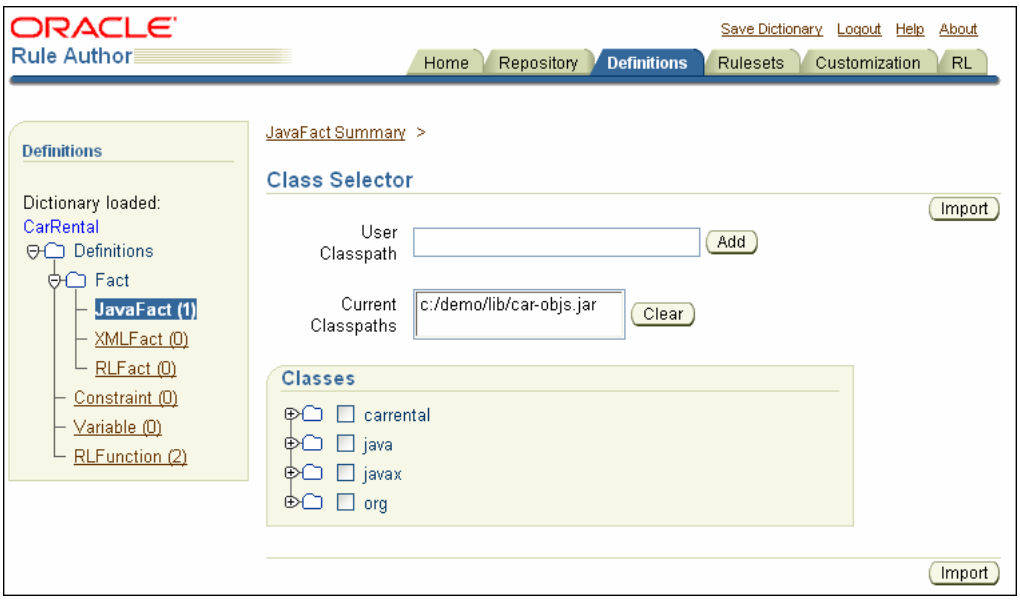


5. Click **Create**. This shows the Class Selector page.
6. On the Class Selector page, the **User Classpath** field lets you add a classpath. For example, for the Java How-To enter the following in the **User Classpath** field.

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

   Replace $*HowToDir* with the directory where you installed the Java 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 about adding Java classes and packages to Rule Author

2.5.3 Importing Java Classes to a Data Model

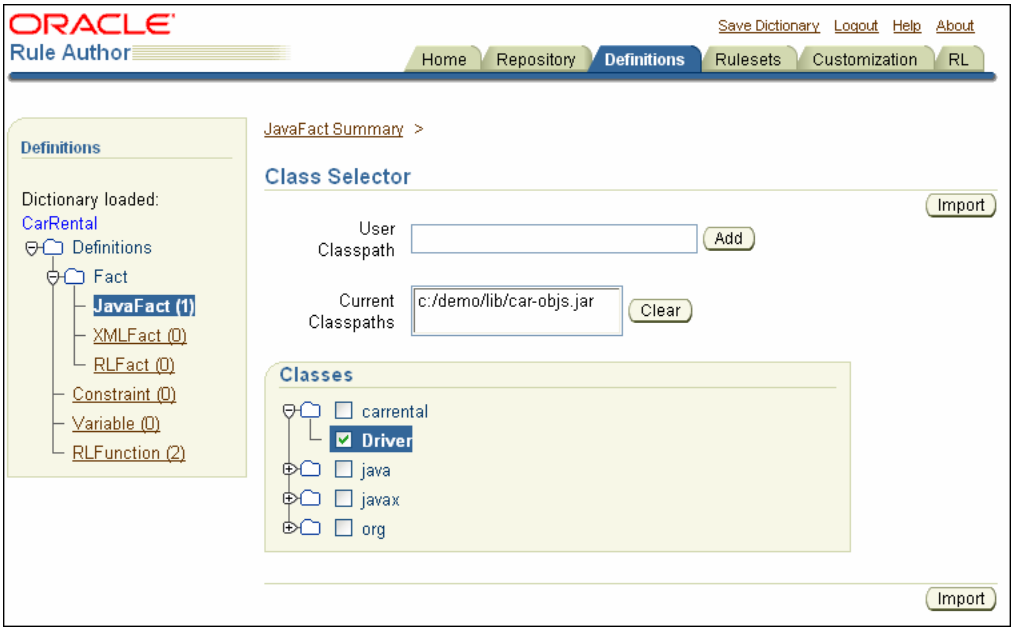
Next, you must select the Java classes that you want to import into the data model from the Class Selector page. To add the `Driver` class to the data model, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. In the navigation tree, click the JavaFact folder.
3. On the JavaFact Summary page, click **Create**. 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**. After you 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 imported 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:

- In 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.
- The Class Selector page includes the Classes box, which shows the Java classes available from the current classpaths.
- 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).

- 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 want to assign an `ArrayList` to a variable of type `Object`, you must import `ArrayList`, `AbstractList`, and `AbstractCollection` into the data model. Otherwise, type-checking does not work correctly and expressions do not validate.
- On Windows systems, you can use a backslash (\) or a slash (/) 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 Java How-To.
- To improve performance the classes box navigation tree. 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.
- Classes and interfaces that you use in Rule Author must adhere to the following rules:

  a. If you are using a class or interface and its superclass, you must declare the superclass 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".
- In the **User Classpath** you can specify a jar file, a zip file, or a full path for a directory.
- 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, to import a class `cool.example.Test1` located in `c:\myprj\cool\example\Test1.class` to the data model, specify the **User Classpath** value, `c:\myprj`.
- Do not use RL Language 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 you are 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 the 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. Rule Author provides an alias feature to allow you to refer to business objects, or facts, in rules using a vocabulary that is intended for business people. 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 appear 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. In the navigation tree, click the **JavaFact** node 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 table for the `carrental.Driver` class.
4. At the top of the page, under the **Name** field, in the Alias box, enter the alias `DriverData` in the **Alias** field for the class `carrental.Driver`.
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 click **OK** or **Apply**, Rule Author does not save your changes.

See Also: "Viewing Java Objects in a Data Model" on page 3-9

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. In the navigation tree, click `RLFunction` in the Definitions folder 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 Arguments Area.

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

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. In the navigation tree, click the **JavaFact** node 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 table for `carrental.Driver`.
4. For each entry in the Properties table, specify the desired visibility using the Visible check box. For this example, only the member variables `age` and `name` must be visible.
5. Click **OK** to save your changes and return to the JavaFact Summary page.
6. Save the dictionary.

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

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

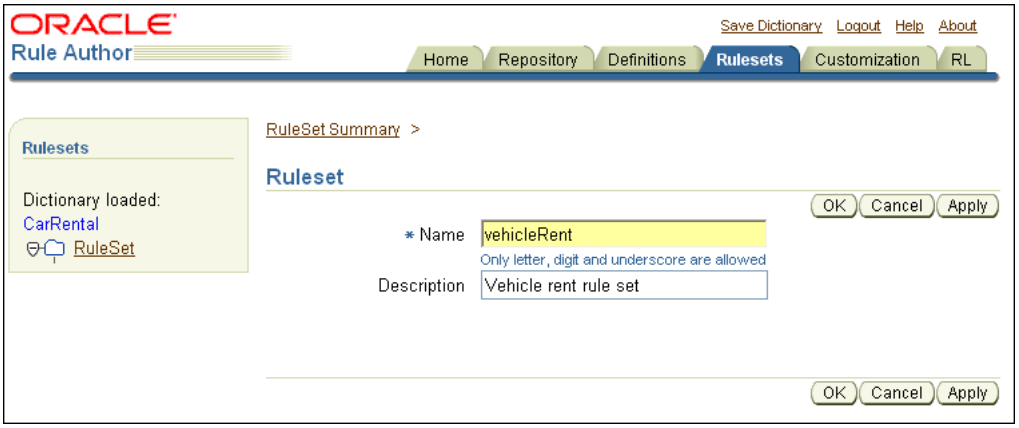
Before you can create a rule using Rule Author, you must create a rule set. A rule set is a container for rules.

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.
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. After you click **OK**, the new rule set appears in the navigation tree under RuleSet.
7. Save the dictionary.

Note: 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 check box 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 create a rule called UnderAge. This rule tests for the following:

If the age of a driver 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 age of the driver.
- Retract the matched driver object. 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 or if the action associated with the rule changes the state, so that the fact must be retracted to represent the current state of the Rules Engine.

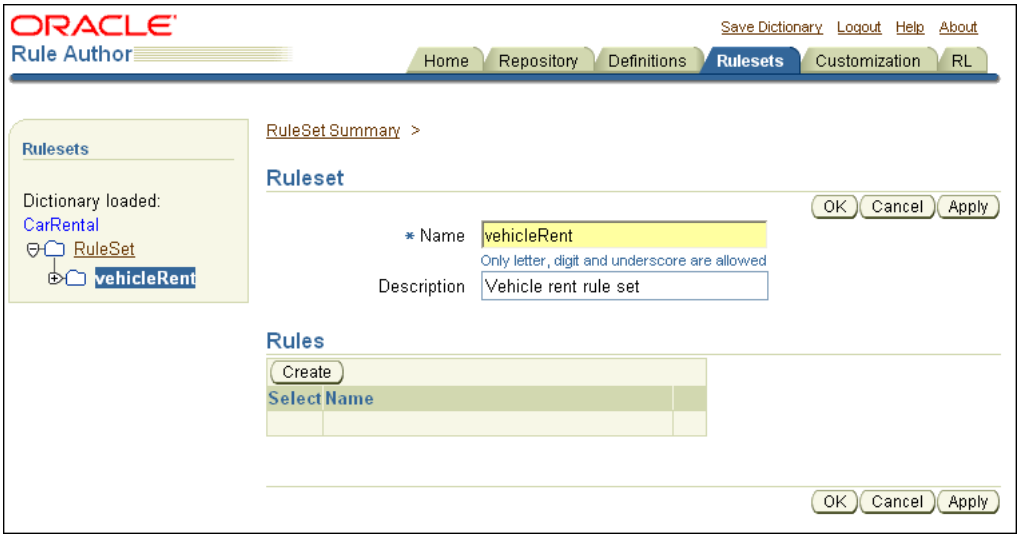
2.7.2.1 Adding the UnderAge Rule for the Car Rental Sample

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 2-14.
2. Click the vehicleRent folder in the tree. This displays the Ruleset page, with a table listing rules (see Figure 2–13).

> **Note:** If you have not previously created other rules, then the Rules table is empty.

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

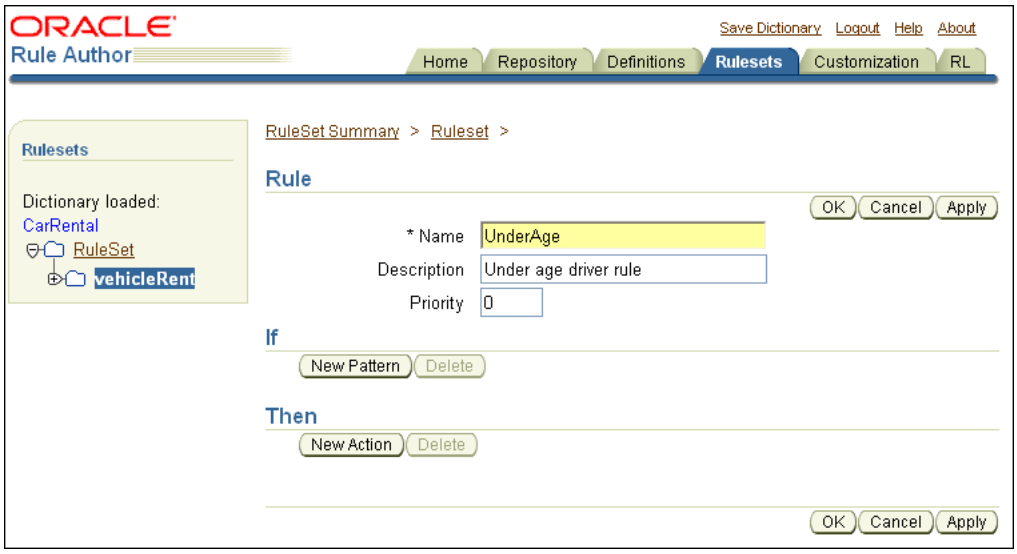


3. Click **Create**. This displays the Rule page.
4. Enter `UnderAge` in the **Name** field.
5. Do not change the default value, 0, in the **Priority** field.

> **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.

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

Figure 2–14 Rule Author Rule Page



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

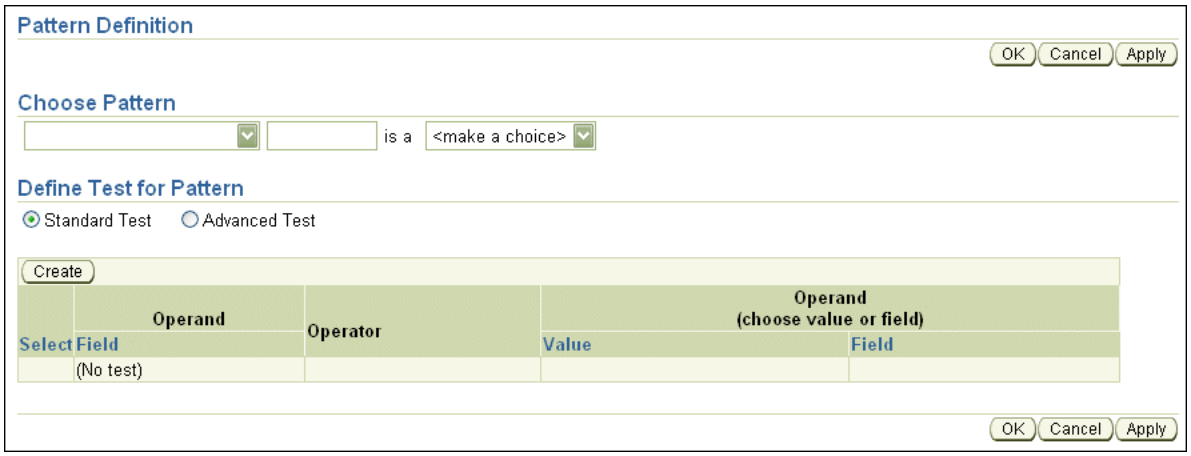
2.7.2.2 Adding a Pattern to the UnderAge Rule

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

1. In the **If** box on the Rule page, click **New Pattern**. This displays the Pattern Definition page, which contains two areas: Choose Pattern and Define Test 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 that 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 that 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 UnderAge Rule" on page 2-20

2.7.2.3 Defining Tests for Patterns with the UnderAge Rule

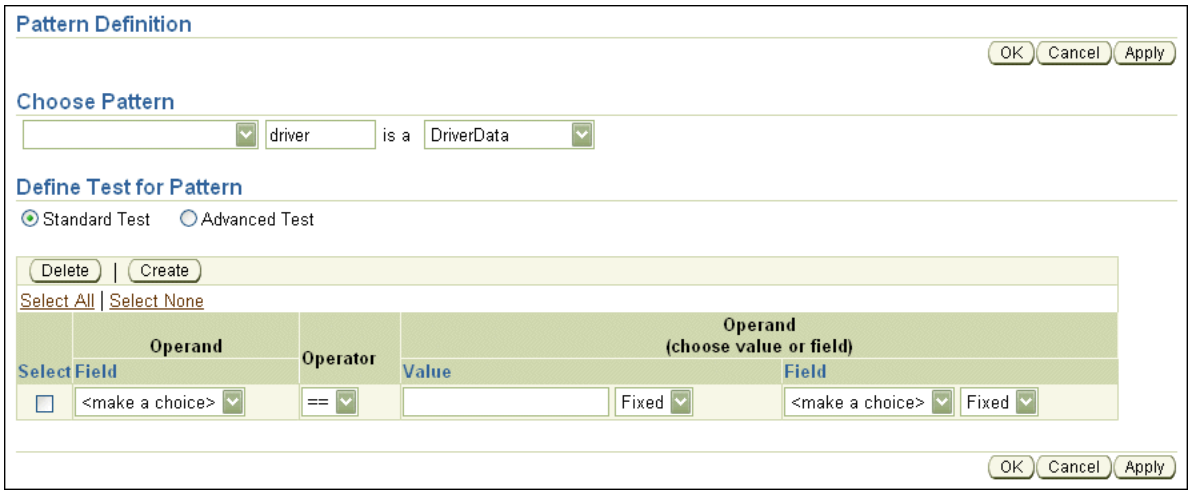
To add a test for a pattern, do the following:

1. From the **Rulesets** tab, in the navigation pane click the rule that you want to add a test for. For this example, click the node in the navigation pane for the UnderAge rule. This displays the Rule page.

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

3. On the Pattern Definition page, select the **Standard Test** button, then click **Create** (see Figure 2–16). For more information, see Section 3.8.1, "Using the Advanced Test Expression Option".

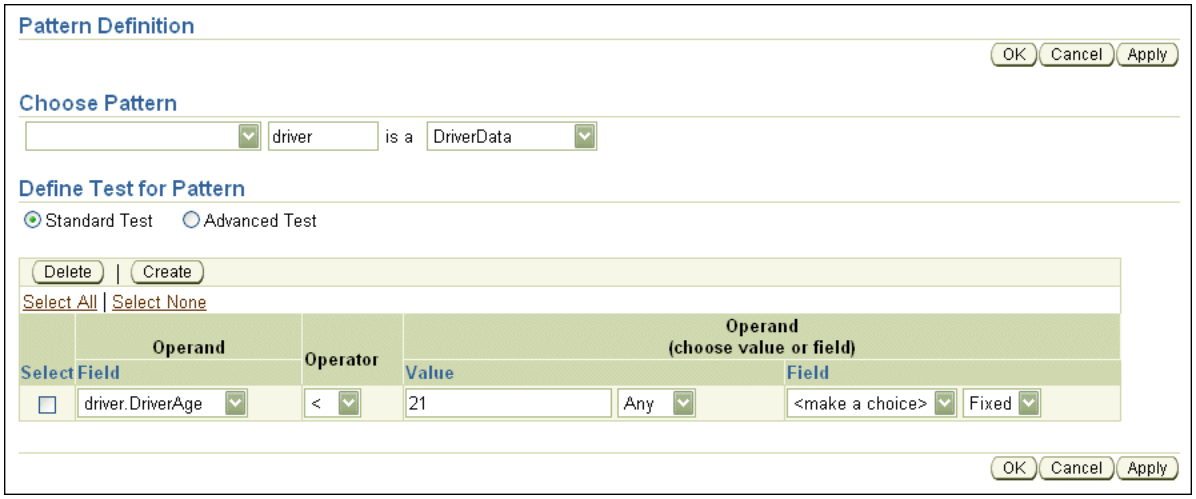
Figure 2–16 Rule Author Rule Pattern Definition Page with Define Tests for Pattern Fields



4. In the **Operand** column, from the first **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 values `Any` and `Fixed` (see Figure 2–17). These values are called constraints, and they enable or disable customization. Use `Fixed` to make the field read-only, which specifies no customization for the field. Select `Any` to specify that Rule Author allows changes to the value (this allows nontechnical users to make modifications to the field from the Customization tab).

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

Figure 2–17 Rule Author Pattern Definition Page with Values for the UnderAge Rule



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

> **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.

10. Save the dictionary.

Note the following when you define a test for a pattern:

- In **Standard Test** expressions, the tests only evaluate to true when all of the tests that you define match. Additionally, in standard tests, no grouping is allowed, and functions with parameters are not allowed. However, with standard tests you can define constraints for customization.
- In **Advanced Test** expressions, the tests do not have the restrictions of standard tests, but they do not allow the use of constraints. Advanced test expressions are not directly saved using RL Language because Rule Author incorporates aliases in the expressions; aliases are not supported in RL Language (Rule Author maps aliases to variable names).

See Also:

- "Customizing Rules for the Car Rental Sample" on page 2-23
- "Working with Constraints" on page 3-2

2.7.2.4 Adding Actions for the UnderAge Rule

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

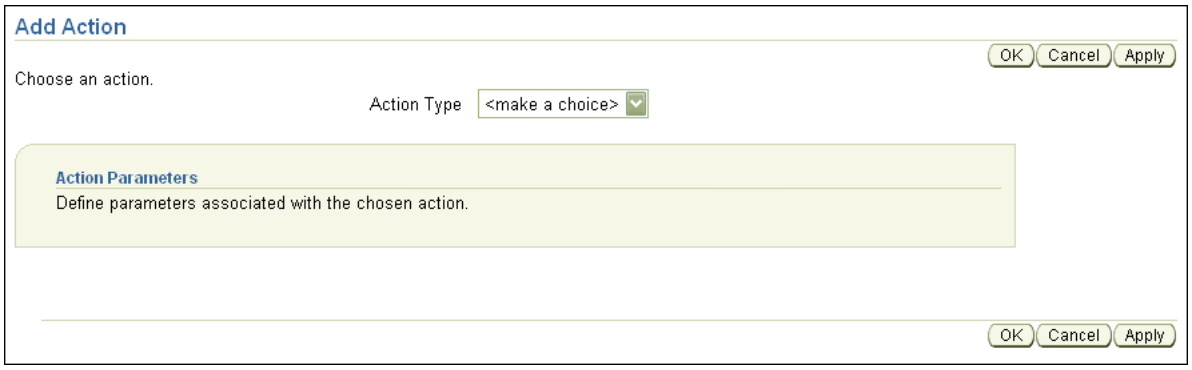
In this section, you add two actions for the UnderAge rule. The first action prints the result. The second action retracts the driver fact from Rules Engine. You might want to retract a fact for a number of reasons, including:

- If you are done with the fact, and you want to remove it from Rules Engine.
- If the action associated with the rule changes the state, so that the fact must be retracted to represent the current state of 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, under the vehicleRent folder, click the node for the UnderAge rule.
3. On the Rule page in the **Then** section, click **New Action**. This displays the Add Action page (see Figure 2–18).

Figure 2–18 Rule Author Add Action Page



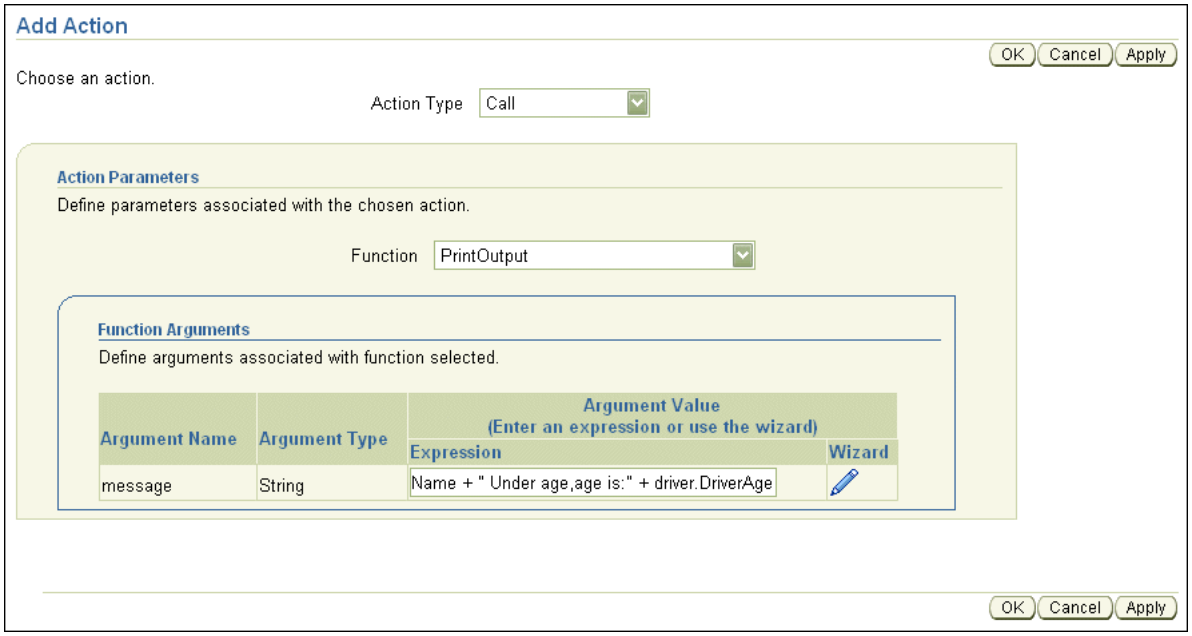
4. From the **Action Type** box, select the `Call` item. This shows the Action Parameters box.
5. From the **Function** list, choose `PrintOutput` (if you did not define an alias for this function, then this is `DM.println`). This expands the Function Arguments box.
6. In the Function Arguments box,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

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

Note 1: Rule Author uses a Java-like syntax for expressions. 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 display 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, because the Wizard presents a variable selection box.

Figure 2–19 Rule Author Add Action Page for the UnderAge Rule



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

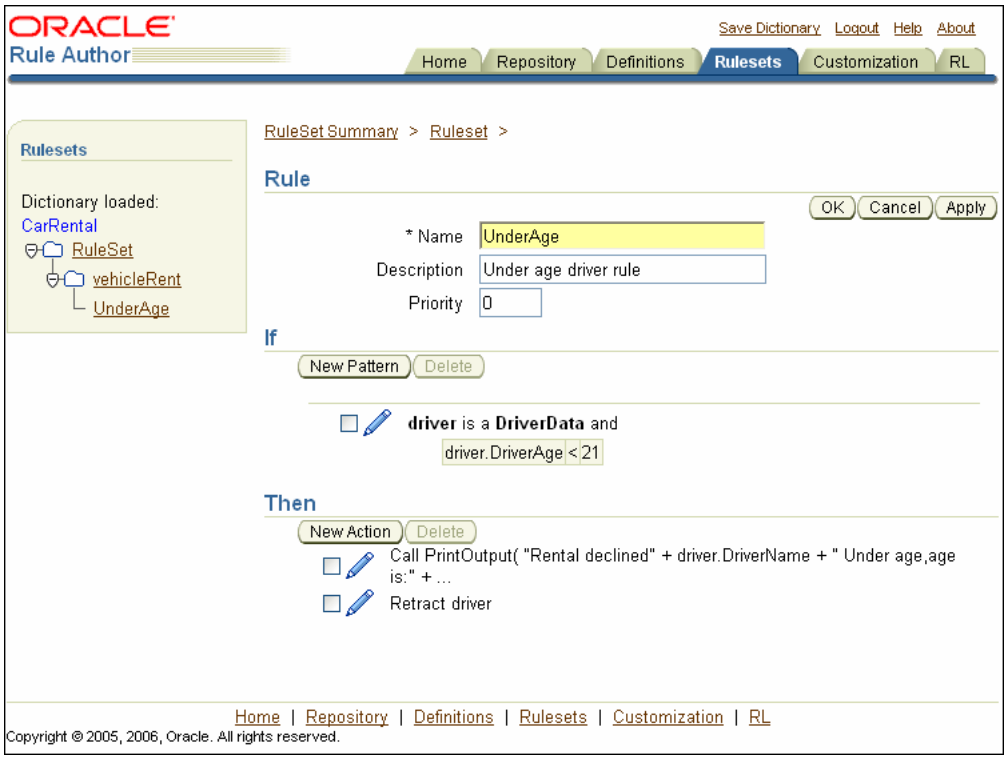
Next, add the retract action for the UnderAge rule. Perform the following steps to add this second action for the rule:

1. Click the **Rulesets** tab.
2. Under the vehicleRent folder, click node for the UnderAge rule.
3. On the Rule page, from the **Then** box, click **New Action**. This displays the Add Action page.
4. From the **Action Type** box, select `Retract`. This shows the **Action Parameters** box.
5. From the Action Parameters area, in the **Fact Instance** list, select `driver`. The pattern name `driver` refers to the single instance that was matched by the pattern.
6. Click **OK** to save your changes and return to the Rule page.
7. Click **OK** on the Rule page (see Figure 2–20).

> **Note:** Changes made to the action 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 action definition.

8. Save the dictionary.

Figure 2–20 Rule Author UnderAge 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 Table 3–2 for more information on Action Types.

See Also: *Oracle Business Rules Language Reference Guide*

2.8 Customizing Rules for the Car Rental Sample

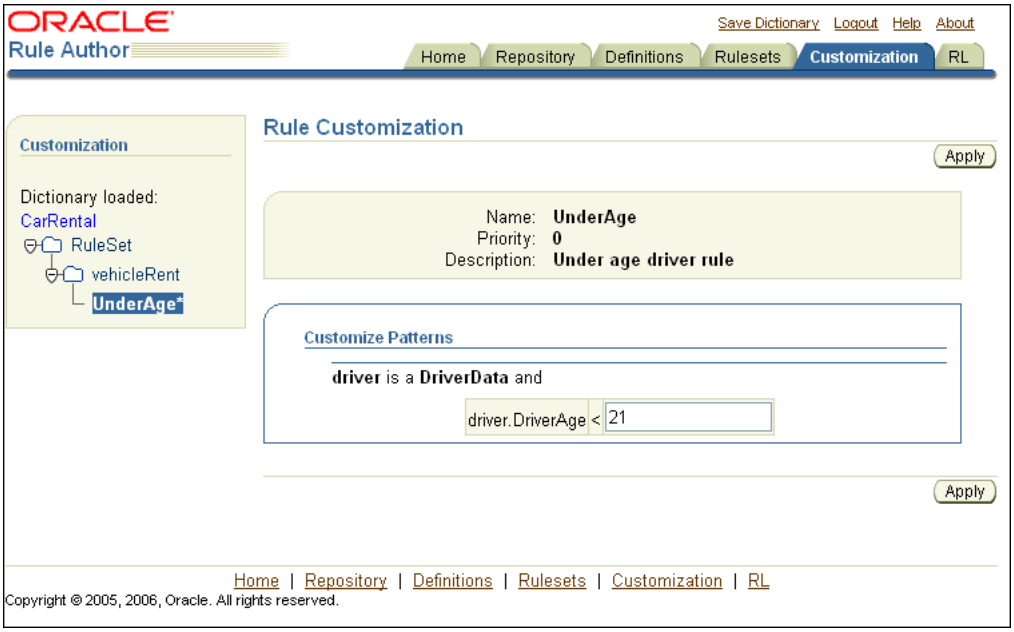
The Rule Author **Customization** tab is designed for business users. Rule developers use the **Allowed Values** field on the Pattern Definition page, which is available from the **Rulesets** tab, to specify if customization is allowed. When customization is allowed, you can specify a range of valid values for the customizable field. 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 underage 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 node for the UnderAge rule followed by an asterisk (*), which indicates that the rule is customizable.
2. In the tree, click the node for the UnderAge rule (see Figure 2–21).

Figure 2–21 Rule Author Rule Customization Page for the UnderAge 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` 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 Java How-To.

> **See Also:** "Defining Tests for Patterns with the UnderAge Rule" on page 2-18 for information about 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 topics:

- Importing the Rules SDK and Rules RL Language Classes
- Initializing the Repository with Rules SDK
- Loading a Dictionary with Rules SDK
- Specifying a Rule Set and Generating RL Language 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, where $*HowToDir* is the directory where you installed the Java How-To.

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 should be 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 Language Classes

The first step when you write a rule-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.sdk.store.jar.Keys;

import oracle.rules.rl.RuleSession;

import carrental.Driver;
```

2.9.2 Initializing the Repository with Rules SDK

When building a rule-enabled Java application, perform the following steps 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. Example 2–2 shows a file type repository.
3. Create a repository instance using the `RepositoryManager.createRuleRepositoryInstance` method.

4. Define a `RepositoryContext` and set appropriate properties. For a file repository, this step specifies 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";

RepositoryType jarType =
     RepositoryManager.getRegisteredRepositoryType( Keys.CONNECTION );

RuleRepository repo = RepositoryManager.createRuleRepositoryInstance( jarType );

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

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

To load a WebDAV repository instead of a file repository as shown in Example 2–2, you should use the `getWebDAVRepository` method. The comments include a sample using this method in the program `TestMain.java` in the $*HowToDir*`/src/carrental` directory, where $*HowToDir* is the directory where you installed the Java How-To.

2.9.3 Loading a Dictionary with Rules SDK

When you build a Java application that uses rules you must 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" );
```

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

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

In Rule Author, a rule set includes two components, a data model, which is global and applies to 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 Rules Engine and maintains the state of Rules Engine across a number of rule executions. A `RuleSession` object is the interface between the application and Rules Engine.

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

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

> **Note:** The order of the `executeRuleset()` calls is important. You must 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, 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 normally assert some 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 the 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 *$HowToDir*`/lib` directory includes `TestMain.jar`, a ready-to-run Oracle Business Rules Java application that uses the `CarRental` dictionary. If you change the dictionary name, then you must modify `TestMain.java` (the source is available in the directory *$HowToDir*`/src`, where *HowToDir* is the directory where you installed the Java How-To).

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

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

Example 2–8 Sample Run of the Car Rental Program

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

Note that not all the facts match the rules and produce output. The example shows the output for the driver, an asserted fact that matches the UnderAge rule.

3

Working with Rule Author Features

This chapter describes how to use some of the more advanced Rule Author features.

The chapter includes the following sections:

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 How-To you built in Chapter 2. In 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:

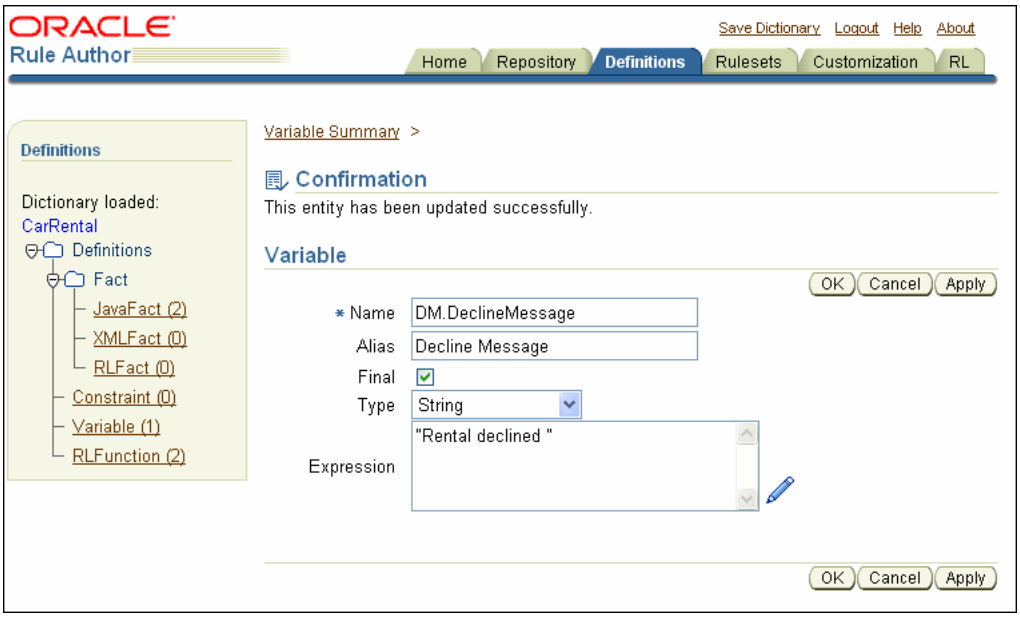
1. Click the **Repository** tab and load the CarRental dictionary.
2. Click the **Definitions** tab.
3. In the navigation tree, click the **Variable** node. The Variable Summary page shows a table with a row that includes the text, `(No items found)`, this indicates no variables are defined.
4. Click **Create**. This shows the Variable page.
5. In the **Name** field, enter `DeclineMessage`.

6. In the **Alias** field, enter `Decline Message`.
7. Select the **Final** check box (by default this box is selected).
8. In the **Type** box, select `String`.
9. In the **Expression** box, enter `"Rental declined "`.

   To use the Wizard to assist you with creating an expression, click the edit icon to display the Wizard.
10. Click **Apply**. Rule Author shows a confirmation message (see Figure 3–1).

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).

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 use only variables specified as final variables in the test for a rule (nonfinals are not allowed).

  See Also: "Defining Tests for Patterns with the UnderAge Rule" on page 2-18 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 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 the regular expression definition in Java.

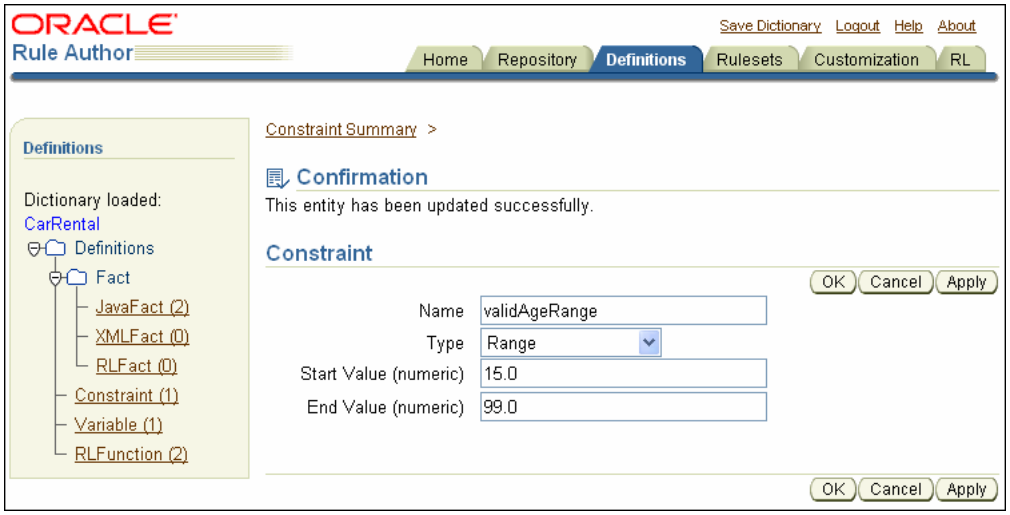
Note: The regular expression constraint requires quotation marks around strings.

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. In the navigation tree, click the **Constraint** node. The Constraint Summary page shows a table that indicates that no constraints are defined.
4. Click **Create**. This shows the Constraint page.
5. In the **Name** field, enter `validAgeRange`.
6. From the **Type** list, select `Range`. This updates the Constraint page display to show 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



Next, use this constraint by adding `validAgeRange` 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. Select the **Rulesets** tab.
3. In the tree, select the node to show the UnderAge rule.
4. In the **If** box, select the edit icon. 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 only lets you enter values from the specified range and rejects invalid entries.

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.

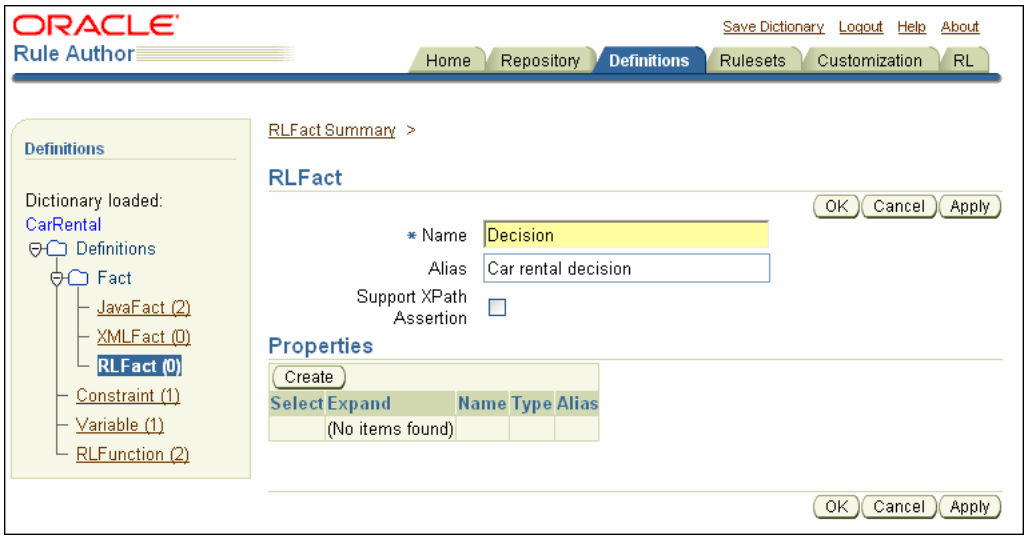
See Also: "Defining Tests for Patterns with the UnderAge Rule" on page 2-18 for information about using the Allowed Values field to use constraints

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:

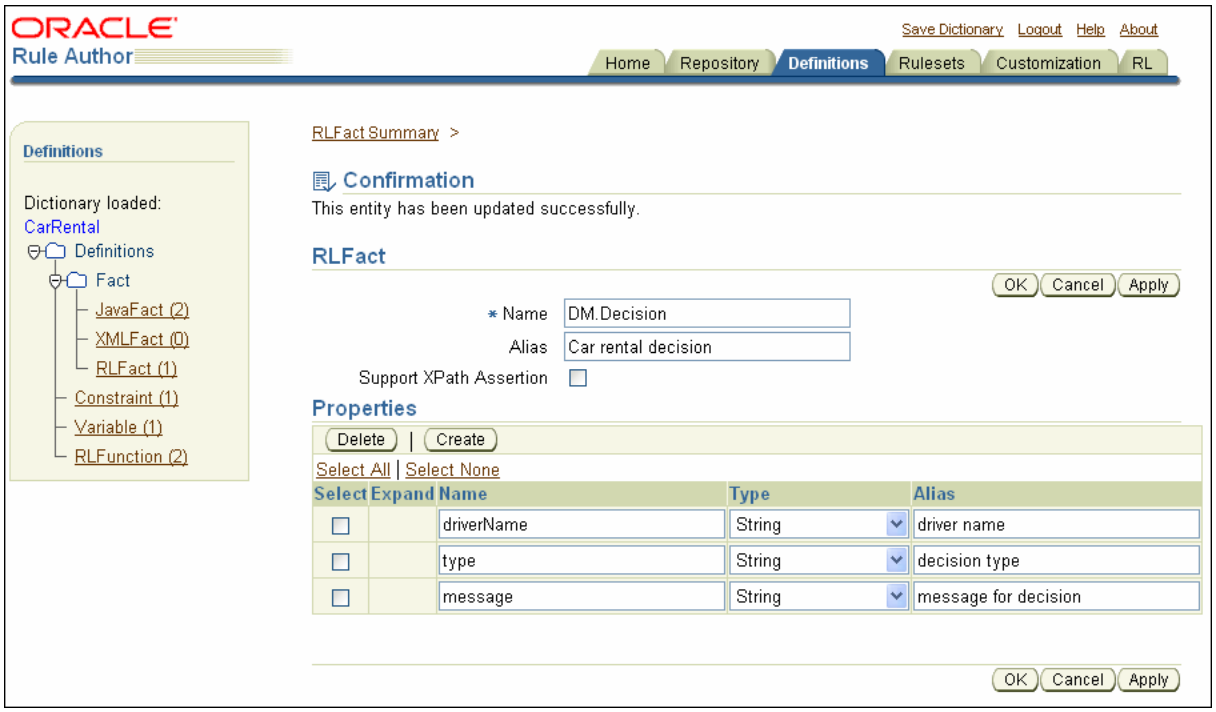
1. Click the **Repository** tab and load the CarRental dictionary.
2. Click the **Definitions** tab. In the navigation tree under Facts, click the **RLFact** node. This displays the RLFact Summary page that shows that no RLFacts are defined.
3. Click **Create**. This shows the RLFact page.
4. Enter `Decision` 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**. Rule Author 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 Lists" on page 3-10 for information about 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 you defined in Section 3.3.

Note 2: For RL Language generated from the Rules SDK, for example, using Rule Author to create rules, global variables may not be referred to directly in an RL Language 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. Select the **Definitions** tab.
3. In the navigation tree, select **RLFunction**. This shows the RLFunction Summary page.
4. Click **Create**.
5. Enter `showDecision` in the **Name** field.
6. Enter `Show Decision` in the **Alias** field.

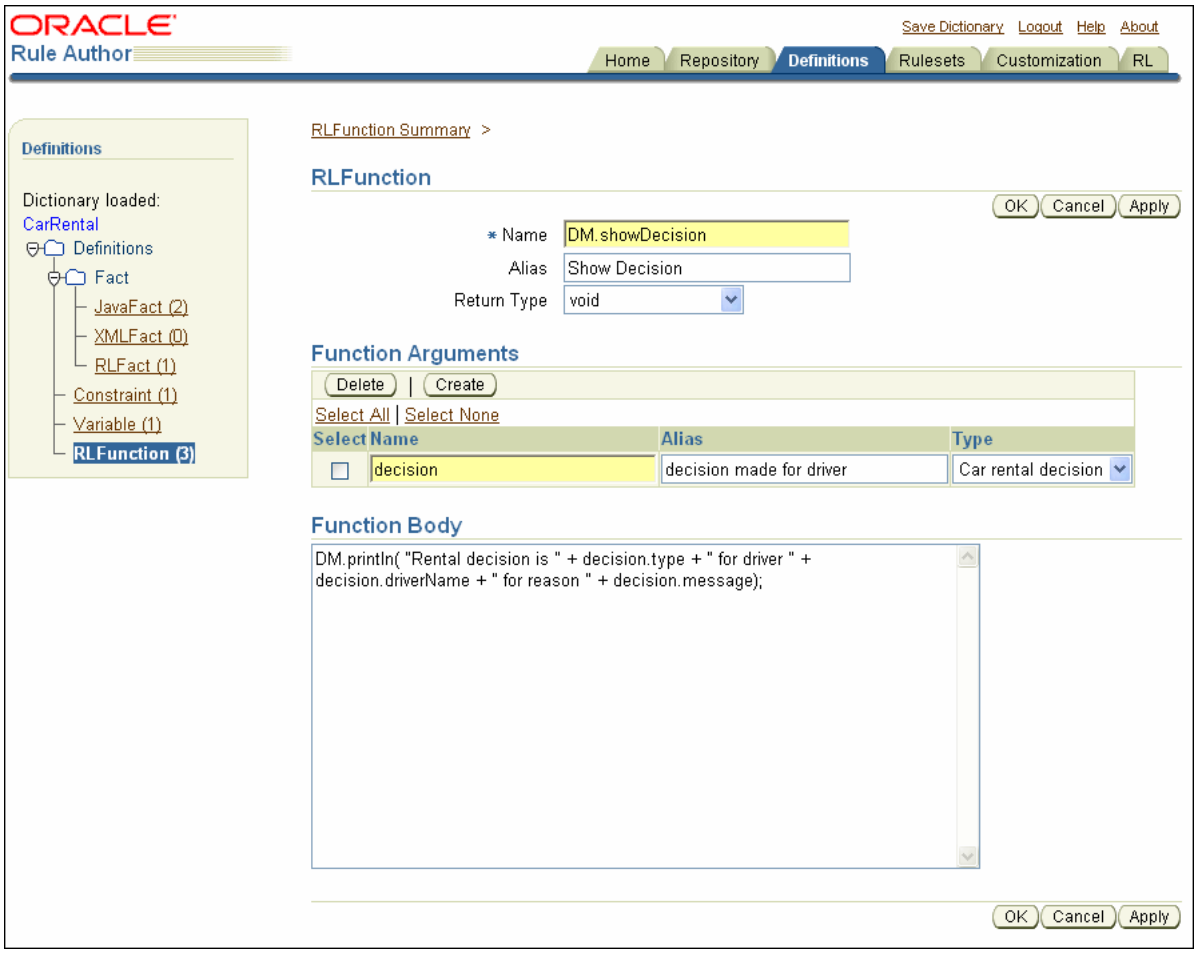
> **Note:** If you are defining a function in Rule Author, you must specify a valid alias in the **Alias** field, even though the actual function name (not the alias) must be used in the function body.

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. In the left navigation pane, click the **RLFunction** node. 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 must create a new rule that checks for Decision facts and provides an action, using the `showDecision` function, to show the results.

3.5 Working with Rules

When you add actions to a rule, the actions are associated with pattern matches. When the "If" portion of a rule matches, the rules engine activates the "Then" portion and prepares to run the actions associated with a rule. Table 3–2 shows the types of actions that you can use when you create a rule.

Table 3–2 Action Types

Action Type	Description
Assert	Asserts a fact used in a pattern. If a fact matched in a pattern is changed, that fact must be asserted again to notify the rules engine that the fact has changed
Assert New	creates a new fact type instance and asserts that instance to the rules engine.

Table 3–2 (Cont.) Action Types

Action Type	Description
Assign	assign a value to a variable or fact property. If a new value is assigned to a fact property, the fact must be asserted again in order for the rules to be re-evaluated with the new value.
Call	Allows you to call a function to perform some action.
Retract	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 Rules Engine. If the action associated with the rule changes the state, so that the fact must be retracted to represent the current state of Rules Engine.
RL	Create free form RL text that will be executed directly. The syntax of this RL is not validated by the SDK, so creating RL actions may result in the generation of invalid RL code from the ruleset.

3.6 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. In the navigation tree, expand the Facts folder and click the **JavaFact** node 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 JavaFact Properties and Methods table.

Table 3–3 and Table 3–4 describe the fields in the JavaFact Properties and Methods tables.

Table 3–3 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 that you can use the class in XPath expressions to assert XML data into a rule session.

Table 3–4 JavaFact Properties and Methods Fields

Field	Description
Visible	Specifies that the property or method appear in Rule Author lists.
Expand	Specifies that the superclass for a property or method that has a superclass appear 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 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 superclass and classes associated through fields and methods to be imported into the data model. The JavaFact Summary page table shows you the superclass and the associated classes for any classes that you import.

3.6.1 Specifying Visibility and Object Chaining for Rule Author 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 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. In the Properties area or the Methods area on the Java Fact page, deselect the property or method to remove it 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).

3.7 Generating Oracle Business Rules RL Language Text

Use the Rule Author **RL** tab to view the RL Language text that represents the data model and the rule sets associated with the dictionary data.

3.7.1 Generating Viewing and Checking RL Language Text

To generate, view, and check the 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 rule set.
5. Click **Check RL Syntax** to validate the RL Language text.

Note: When you use the **Check RL Syntax** feature in Rule Author, if the data model includes any Java classes then the Java classes must be included in the OC4J classpath to perform validation. Also, if the data model includes any XML schema, the generated JAXB class files must be included in the OC4J classpath to perform validation.

Thus, to perform validation you must assure that the OC4J classpath contains the jar files or class for the Java objects, or for the java classes generated from the XML schema.

For Java classes in a JAR file, you can copy the JAR files to 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.

See Also: "Working with Test Rulesets" on page 3-16 for details on using Enterprise Manager to add Java classes as a shared library

3.8 Configuring Rule Author Dictionary Properties

In Rule Author, you can use dictionary properties to specify the default expression type to use in expressions, and to specify logging options.

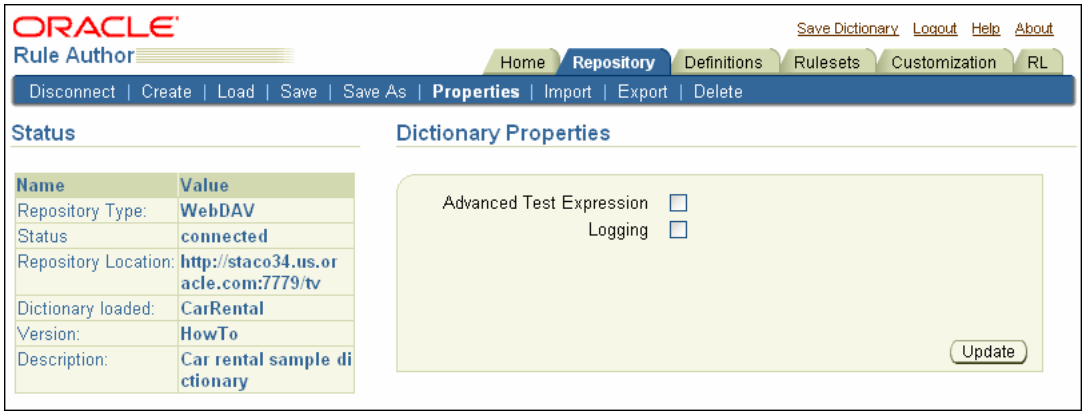
This section covers the following topics:

- Using the Advanced Test Expression Option
- Using the Logging Option

To configure dictionary properties, perform the following steps:

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



Note: Rule Author stores dictionary properties per user. If you change your preferences, and then log out, and then log in as a different user, your dictionary properties reflect the properties for the newly logged in user.

If Rule Author does not use authentication, which is possible if the user configures `web.xml` to disable security, then this also disables saving user level dictionary properties.

3.8.1 Using the Advanced Test Expression Option

The **Advanced Test Expression** check box changes the Rule Author expression mode to advanced for test expressions that are displayed when you edit a pattern for a rule (see Figure 3–7). Tests in a the condition of a rule 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, 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 Test Expression 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 associated with a pattern cannot be changed.

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

Figure 3–7 Pattern Definition Advanced Test Expression Page

3.8.2 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**, the lowest log level generates the least amount of output. **Status** is the medium log level. **Trace** is the highest log level, generating the most 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 specified, 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.

3.9 Deleting a Rule Author Dictionary

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

To delete an individual dictionary version, do the following:

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

   To delete a specific dictionary version, select a dictionary and version in the "Select dictionary version" section, then click **Delete Version**.

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

3.10 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 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 to select the dictionary.

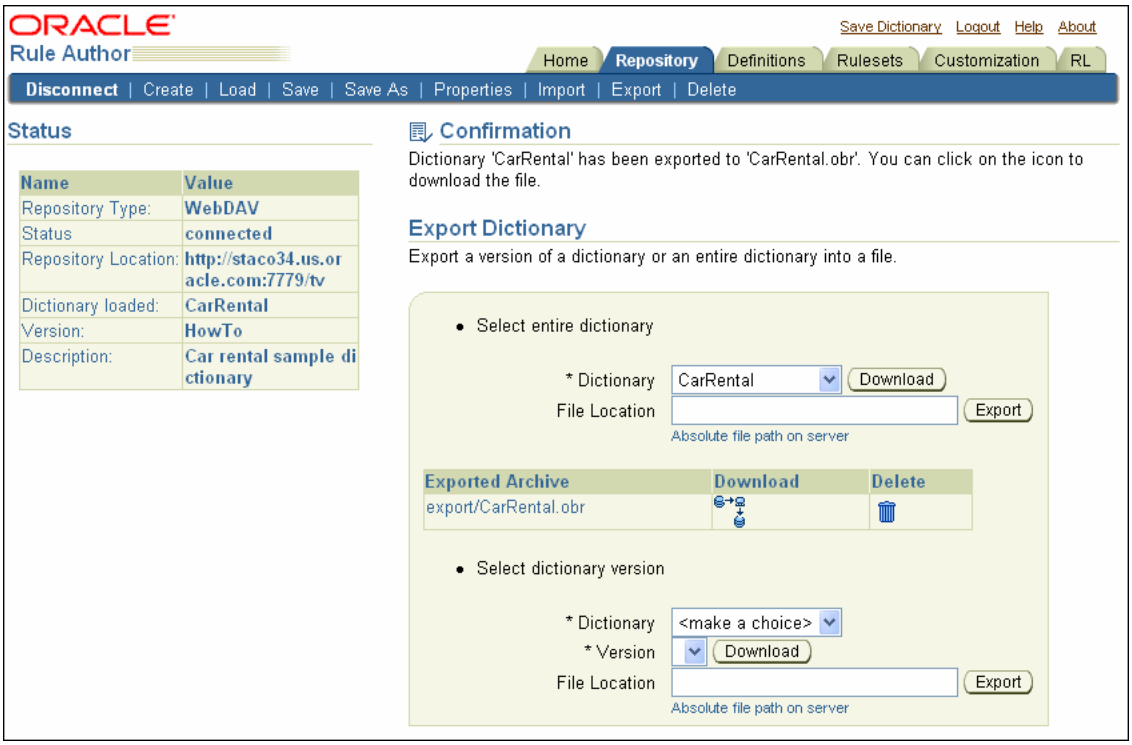
   If the dictionary resides on a different machine (where the Rule Author is not running), you must specify the full path to the dictionary on that server.
3. Click **Import**.

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



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**. This creates a link to the exported archive on the Export Dictionary page. You can then use your browser to download the archive to a location of your choice (see Figure 3–8).

Note: Exporting a dictionary is an incremental operation. Thus, when you export a dictionary version, and you subsequently export another version of the same dictionary to the same jar file, after the export operation, the jar file includes both versions of the dictionary.

For example, if dictionary "dict1" has two versions "v1" and "v2", when the version ("dict1", "v1") is exported to `myExport.jar`, then the jar file contains "v1" content only. If you perform another export of ("dict1", "v2") to the same file `myExport.jar`, after the second export completes the file contains both exported versions of "dict1", "v1" and "v2".

3.11 Working with Test Rulesets

In Rule Author, the Test Rulesets feature allows you to use RL functions to test rule sets. The selected rule sets and their associated data model are executed in a Rule Session and then a user-defined function is called that exercises the rules.

To use the Test Rulesets feature perform the following steps:

1. Create the rule set that you want to test. If the data model includes any Java classes or XML schema, the Java classes or the generated JAXB classes must be included in the OC4J classpath.

   For Java classes, you can put the JAR files in the following directory, and then restart OC4J to add the classes to the classpath:

   ```
   $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 steps:

   a. Navigate to the OC4J home page.

   b. Click the **Administration** tab.

   c. Find Shared Libraries under the **Properties** node and click the icon in the Go to Task column.

   d. Click **Create** and enter the library name and version number, and then click **Next**.

   e. Click **Add**, and then 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 test function, as follows:

   a. Click the **Definitions** tab.

   b. Click the **RLFunction** node in the navigation tree.

   c. Enter `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.

To enable logging, you can call the `watchFacts()` or the `watchActivations()` functions.

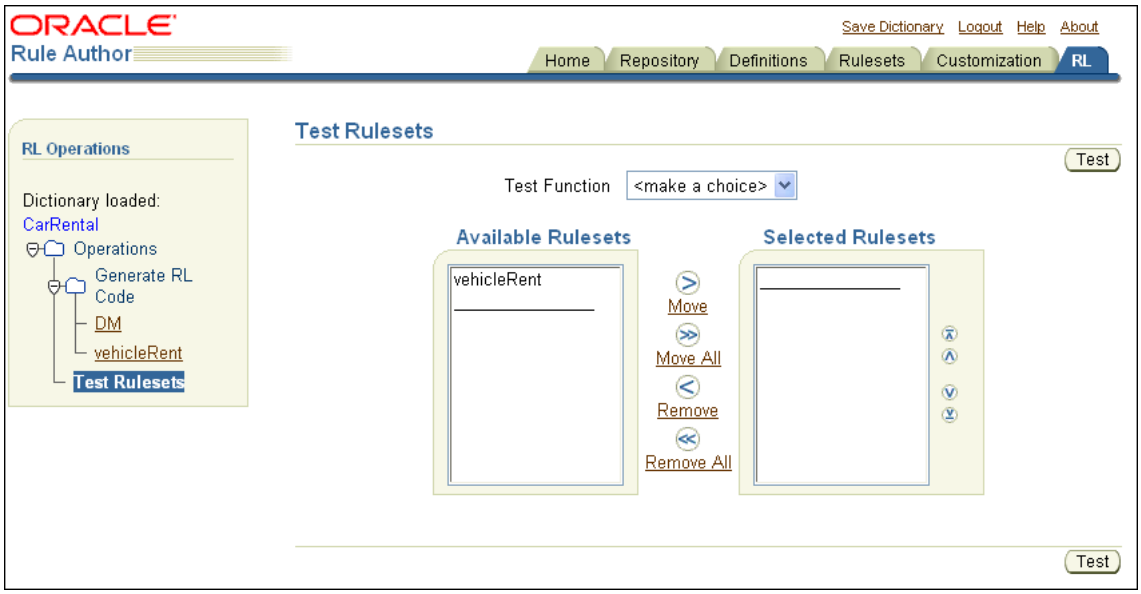
f. Click **OK**. This saves the RLfunction named `DM.test`.

g. Save the dictionary.

3. Click the **RL** tab.

4. In the navigation tree, click the **Test Rulesets** node. This displays the Test Rulesets page.

Figure 3–9 Rule Author Test Rulesets Page

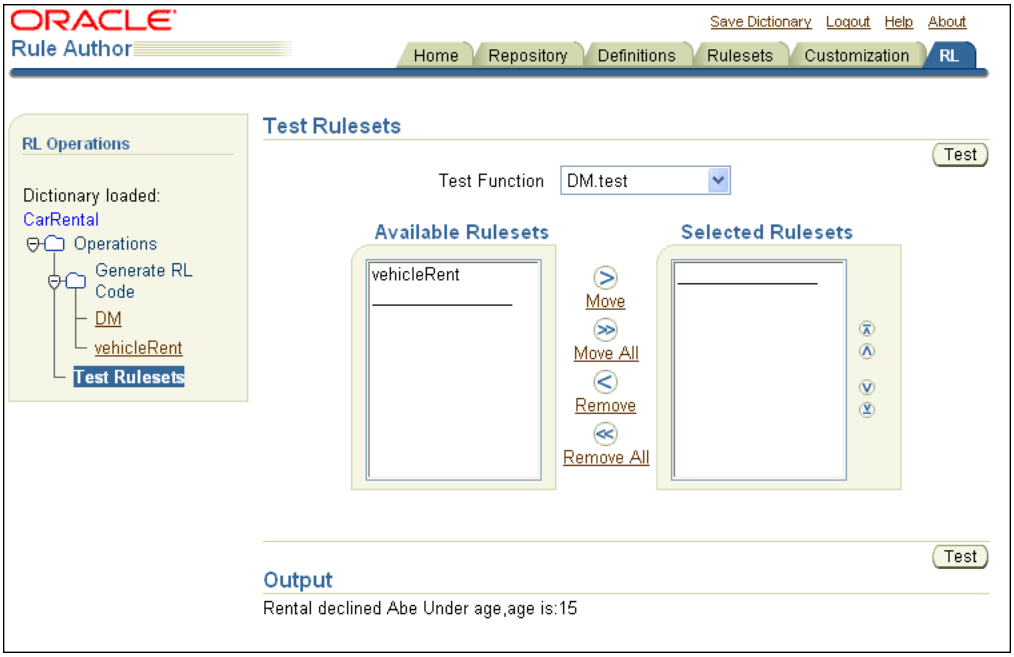


You can see the `vehicleRent` rule set is available. This rule set was created in Chapter 2.

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**.

   This generates RL Language code 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



3.12 Invoking Rules and Obtaining Rules Engine Results

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` method.

This section covers the following topics:

- Overview of Results Examples
- Using a Global Variable to Obtain Results

- Using Container Objects to Obtain Results
- Using Reasoned On Objects to Obtain Results

3.12.1 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 when 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, `sess` is a `RuleSession` object 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 is asserted and only those subscribers registered at that time are notified.

3.12.2 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 those shown in Section 3.12.3 and Section 3.12.4.

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

Example 3–1 Obtaining Results with an 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 the `getSubscribers` function 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));
}
```

3.12.3 Using Container Objects to Obtain Results

In the container objects approach, one or more objects are asserted into working memory to act as a container for results. The RL Language 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 shows the `getSubscribers` function that uses a `java.util.Map` object and adds the subscriber (`key`) and the incident (`value`) are to the `Map` object. The `getSubscribers` function asserts the `Map` object and a `TrafficIncident` object, then invokes the Rules Engine and returns the result in a `Map` object.

Note: The `Map` object is not reasserted even though it has been updated. In general, when an object that is being reasoned on is updated, it should be reasserted. 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 OK not to reassert 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 the `getSubscribers` function and prints out the results.

3.12.4 Using Reasoned On Objects to Obtain Results

In the reasoned on objects approach, one or more objects are asserted into the rules engine working memory and the object references are retained in RL Language (in the `getSubscribers` function). The rule evaluation process 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 an updated `java.util.Set` object containing a list of subscribers that need to be notified. Because the `TrafficIncident` object is being reasoned on, it is reasserted. To avoid an infinite loop of rule firings for the same subscription, 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` object. 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 the code that uses the `getSubscribers` function to assert a `TrafficIncident` object, invoke Rules Engine, and create and return the result `Map` object.

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 the `getSubscribers` function 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 rules XML How-To, driver data is supplied in an XML document that specifies driver information, and the business rules we develop in this chapter determine if a rental company service representative should decline to rent a vehicle due to driver age restrictions (according to rental company business policies that we define).

This chapter includes the following sections:

Note: We call the sample application in this chapter the rules XML How-To. This sample application is available in the Oracle Business Rules area, under the Viewlets and Tutorials heading on the Oracle Technology Web site:

`http://www.oracle.com/technology/products/ias/business_rules/files/how-to-rules-xml.zip`

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.

When you start with an XML schema, using XML documents with Rule Author involves the following steps:

1. Rule Author generates Java classes from the 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. You import XML elements into the data model in a dictionary.
3. You define rules that specify business policies based on the XML elements from an XML document. The process of writing rules for XML documents is very similar to writing rules for Java objects.

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

Note: To use a JAXB binding compiler that is different from the implementation supplied with Rule Author, you can manually perform the XML schema processing using your JAXB binding compiler and then import the generated Java packages and classes into the data model.

For more information about JAXB, see

`http://java.sun.com/webservices/jaxb/`

See Also: "Using Java Objects as Facts in the Car Rental Sample" on page 2-9

4.2 Creating a Rule Author User and 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.

See Also: "Creating a Rule Author User" on page 2-2 for more information

4.3 Creating and Saving a Dictionary for the XML Car Rental Sample

To work with Rule Author, you must start with a dictionary. Rule Author stores rules and their associated definitions in a dictionary. To create or save a dictionary, you must 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 How-To using 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 installed the XML How-To.

This section covers the following topics:

- Connecting to a Rule Author Repository
- Creating a Rule Author Dictionary
- Saving a Rule Author Dictionary with a Version
- Saving a Rule Author Dictionary

4.3.1 Connecting to a Rule Author Repository

In Oracle Business Rules, a dictionary stores rules and the data model associated with the rules.

> **Note:** Regardless of whether you choose to use a WebDAV or file repository, the repository must exist before you can connect to it. See Appendix B, "Using Rule Author and Rules SDK with Repositories" for more information.

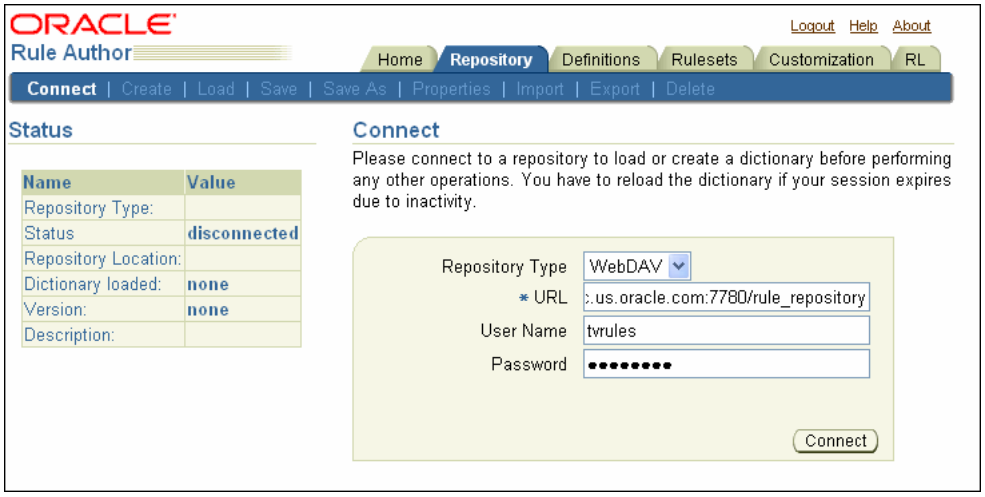
To connect to a repository, do the following:

1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. In the **Repository Type** field select the `WebDAV` repository type.
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 about 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 values in the **Proxy User Name**, and **Proxy Password** fields (as required for the Proxy server).
6. Click **Connect**. After you connect, Rule Author displays a confirmation message.

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.

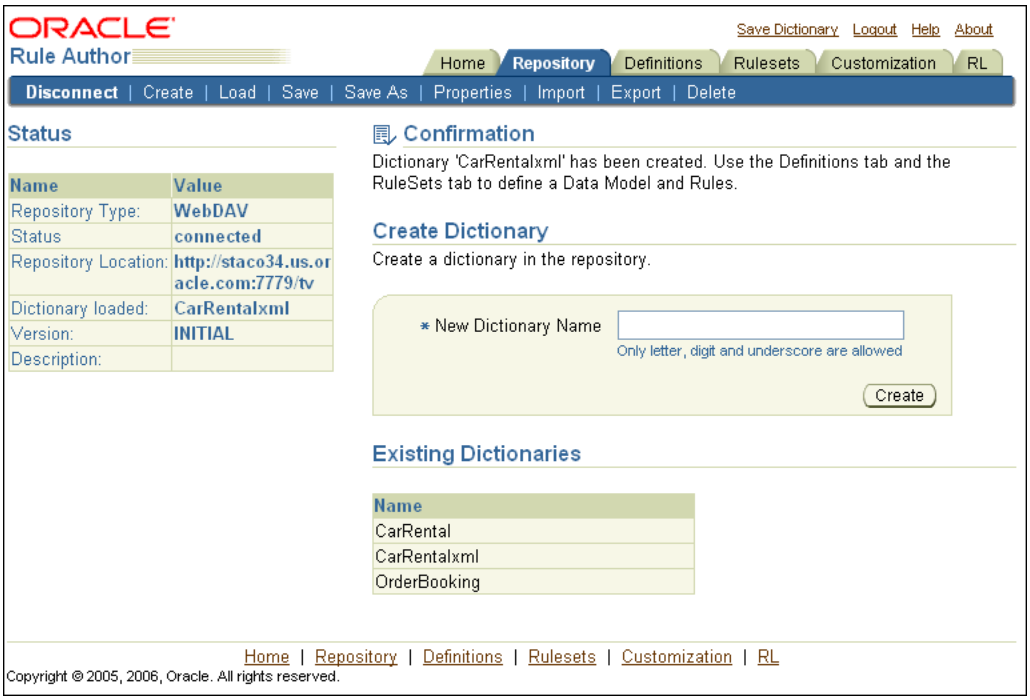
4.3.2 Creating a Rule Author Dictionary

A 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)

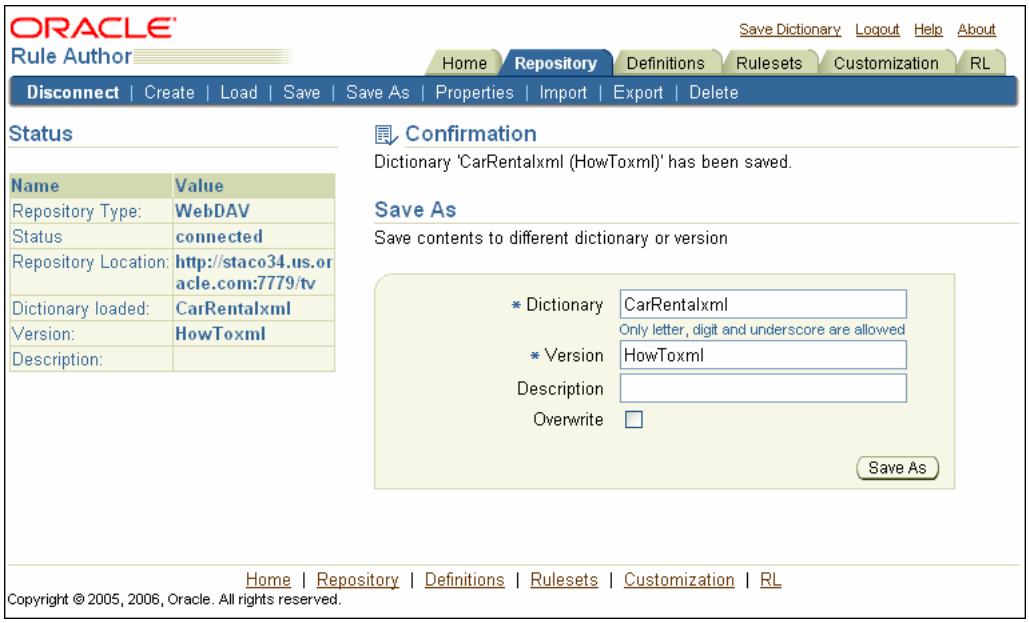


4.3.3 Saving a Rule Author Dictionary with a Version

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. 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**, Rule Author shows a confirmation message (see Figure 4–3).

Figure 4–3 Rule Author Save Dictionary (XML)



4.3.4 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 'CarRentalxml(HowToxml)' has been saved
```

> **See Also:** "Rule Author Session Timeout" on page A-2 for details on configuring the Rule Author session timeout and for details on how Rule Author automatically saves the current work to a dictionary version when a timeout occurs

4.4 Defining a Data Model for the XML Car Rental Sample

Before working with rules you must 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

4.4.1 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. The access the schema definition, replace $*HowToDir* with the directory where you installed the XML How-To.

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

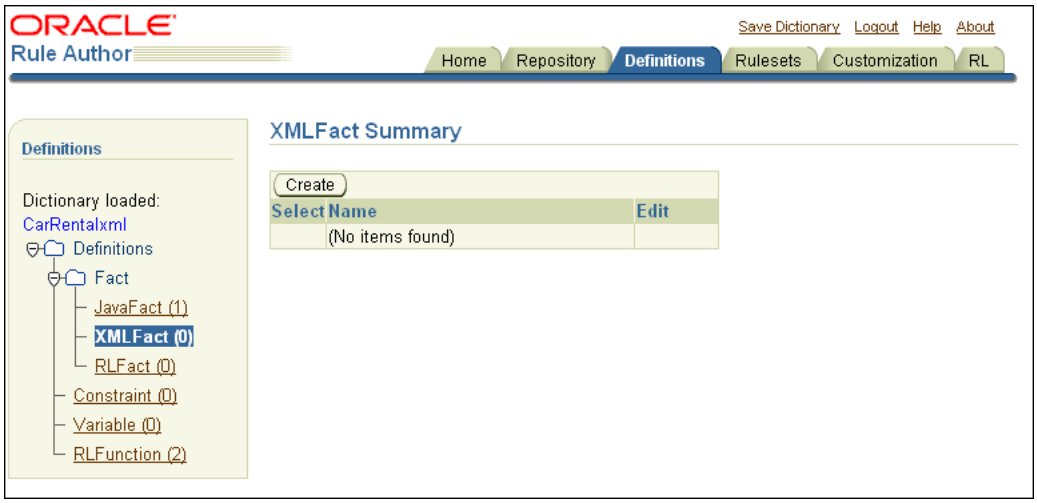
Before you can use XML elements in a data model, Rule Author must generate the JAXB classes representing the XML elements. This step generates the JAXB classes and makes the generated classes and packages associated with the XML schema visible to Rule Author.

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, which 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 XMLFact 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`

   Note: If you choose to access the schema with a URL, and a proxy server is involved, then you must set the following system properties:

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

   For example:

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

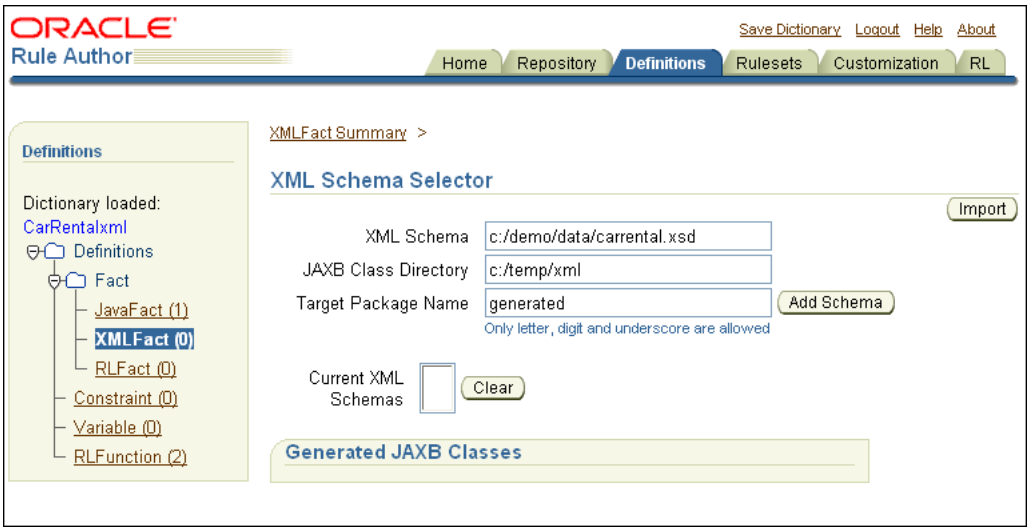
   For more information about setting system properties in an OC4J instance, see *Oracle Containers for J2EE Configuration and Administration Guide*.

   ---

6. In the **JAXB Class Directory** field, enter the directory where you want Rule Author to store the JAXB-generated classes. The directory that you specify must be writable. For example, enter `c:/temp/xml`.
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 target namespace of the XML schema 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, use the name `generated` (see Figure 4–5). Although this example uses the name `generated`, there is nothing special about this name. This name specifies the package for the generated classes.

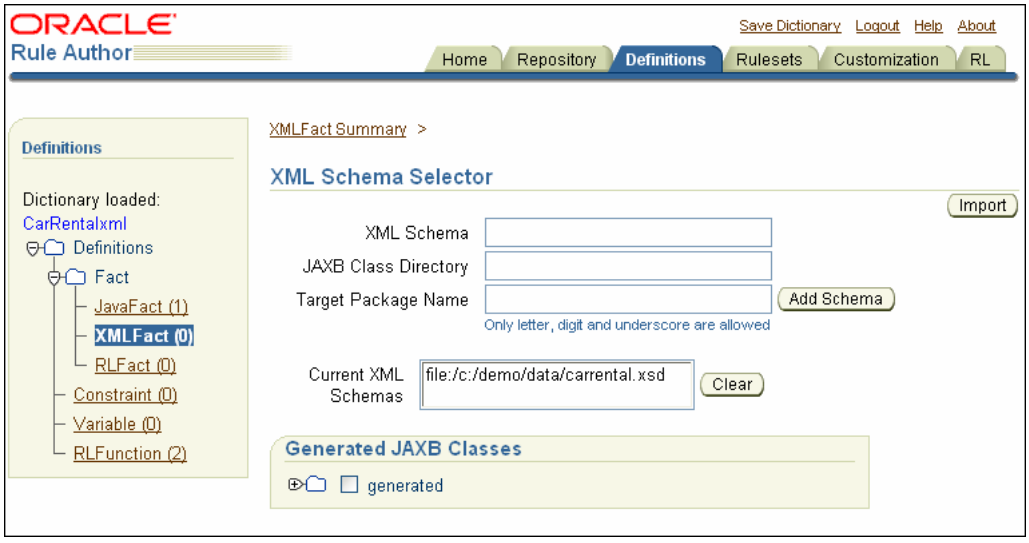
Figure 4–5 Rule Author XML Schema Selection Page



8. Click **Add Schema**.

   This step requires a period of time for Rule Author to process the schema and compile the JAXB, so depending on the size of the schema, you may need to wait for this step to complete. When this step completes the page shows the 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



Note: JAXB sometimes maps XML construct names to different Java identifier names. For example, in JAXB-generated classes, 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.

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

4.4.3 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 XML Schema Selector page from the **Definitions** tab.

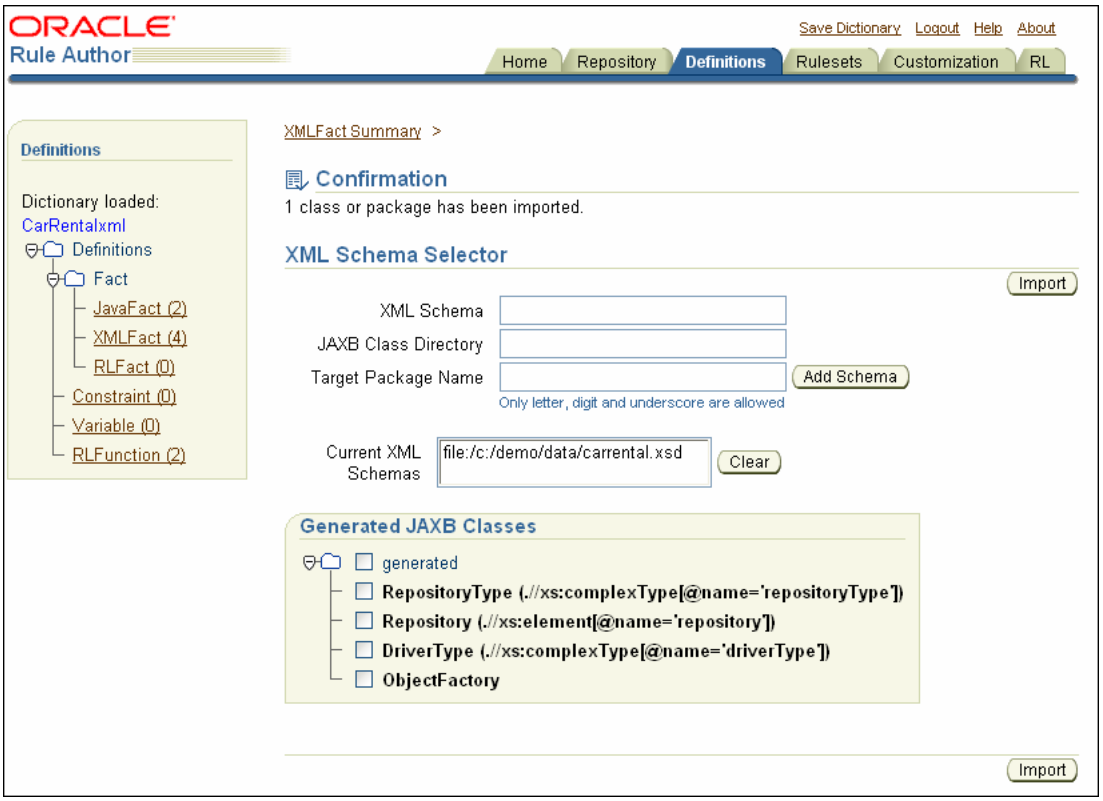
To add `DriverType` from the schema to the data model, do the following:

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".
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:

- In Rule Author, importing an XML fact 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.
- 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 backslash (\) or a slash (/) 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). To import the whole package into the data model, check the package name and click **Import**.
- Do not use RL Language reserved words in Java package names. For more information, see Section D.8, "Using RL Reserved Words as Part of a Java Package Name"
- Oracle Business Rules binds an XML schema to Java classes using the JAXB binding. In most cases, the default bindings generated by the Oracle JAXB binding compiler should be sufficient to meet your needs. There are cases, however, when you may want to modify the default bindings. For example:
  - In cases where the default binding generates a name collision.

– In cases where the default binding generates invalid Java identifiers (it is possible to generate invalid Java identifiers from non-English tag names).

See Also: Java Architecture for XML Binding (JAXB) Specification for details on using custom binding declarations

`http://java.sun.com/xml/downloads/jaxb.html`

4.4.4 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. In the navigation tree, expand the Facts folder and click the **XMLFact** node to display the XMLFact Summary page.

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

Note: Importing a Java class does not cause all of its superclasses 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 Lists" on page 3-10 for details on the **Visible** and **Expand** fields in the XML Fact Properties and Methods table

4.4.5 Saving the Current State of XML Fact Definitions

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

To save the dictionary do the following:

1. Click the **Save Dictionary** link at the top of the page.
2. Click **Save** on the Save Dictionary page.

4.5 Defining the 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. Rule Author provides an alias feature to allow you to refer to business objects, or facts, in rules using a vocabulary that is intended for business people. 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 appear 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
- Specifying the Visibility for Properties and Methods for XML Facts

4.5.1 Specifying the Business Vocabulary for XML Fact Definitions

To specify the business vocabulary for XML fact definitions, do the following:

1. Click the **Definitions** tab to view the Definitions page.
2. In the navigation tree, expand the Fact folder and click the **XMLFact** node to display the XMLFact 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 XMLFact page.
4. At the top of the XMLFact 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, in the **Alias** field, enter `DriverName`.
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:

- The XMLFact 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.

- On the XMLFact 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 XMLFact page. The Expand box is shown in the **Expand** field of the Properties and Methods area. The check box is only shown for methods or properties that include a superclass (Rule Author does not show the **Expand** box for primitive types).
- On the XMLFact page you can specify that properties or classes are not visible in Rule Author lists. Deselect the **Visible** check box to specify that an object is not visible in Rule Author lists (by default objects are visible in lists).
- Make sure the **Support XPath Assertion** box is checked for all XML fact types. For more information, see Section D.10, "XML Facts not Asserted at Runtime".

4.5.2 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. In the navigation tree, click `RLFunction` in the Definitions folder 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 do not change the function arguments alias.

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

4.5.3 Specifying the Visibility for Properties and Methods for XML Facts

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. In the navigation tree, click the **XMLFact** node to display the XMLFact Summary page.
3. Click the edit icon to view the XML fact Properties and Methods for `DriverType` (in the table this entry has the value `DriverData` in the Alias column). This shows the XMLFact page.
4. For each entry in the Properties table and in the Methods table, specify the desired visibility using the **Visible** check box. For this example, only the member variables `age` and `name` need to be visible.
5. Click **OK** or Apply to save the changes.
6. Save the dictionary.

> **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.

4.6 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

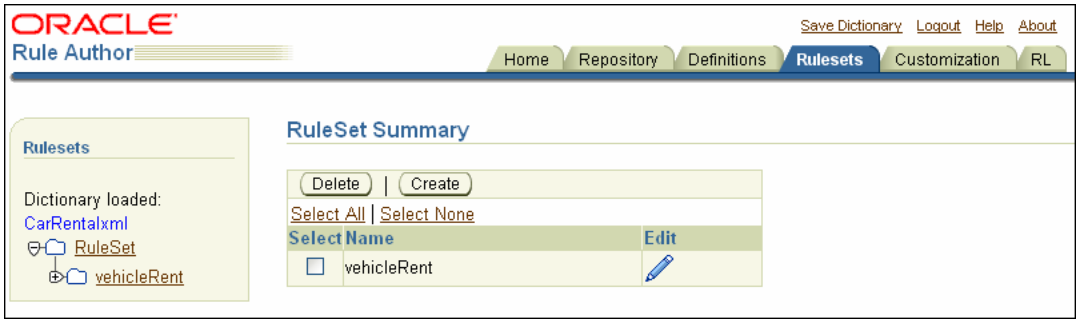
4.6.1 Creating a Rule Set for the XML Car Rental Sample

Before you can create a rule you must 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



4.6.2 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 age of the driver is younger than 21, then decline to rent

The UnderAge rule contains a single pattern for 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 age of the driver.
- Retract the matched driver fact from the rule session. 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 or if the action associated with the rule changes the state, so that the fact must be retracted to represent the current state of the Rules Engine

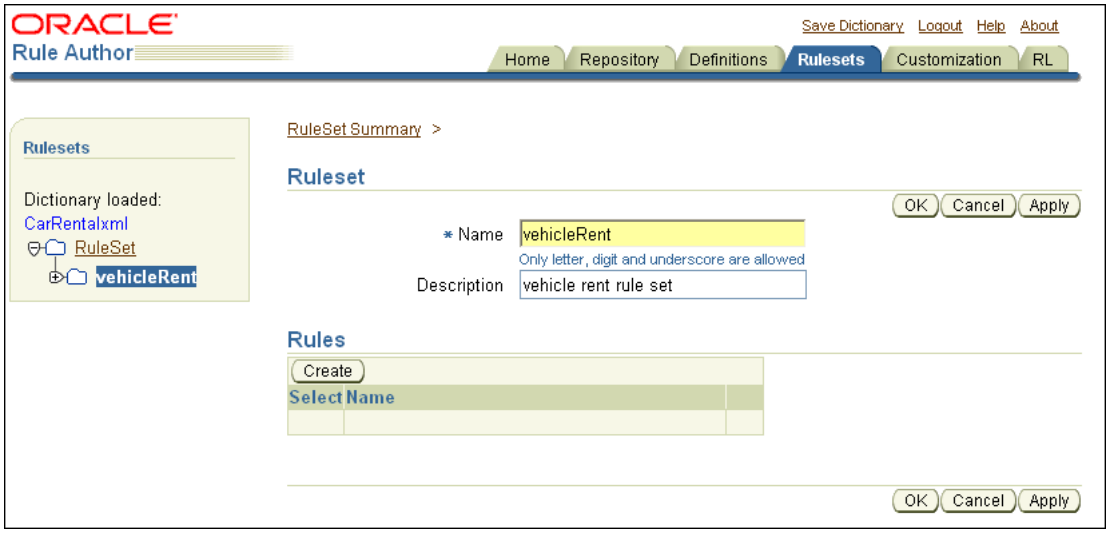
4.6.2.1 Adding the UnderAge Rule for the XML Car Rental Sample

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 4.6.1.
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

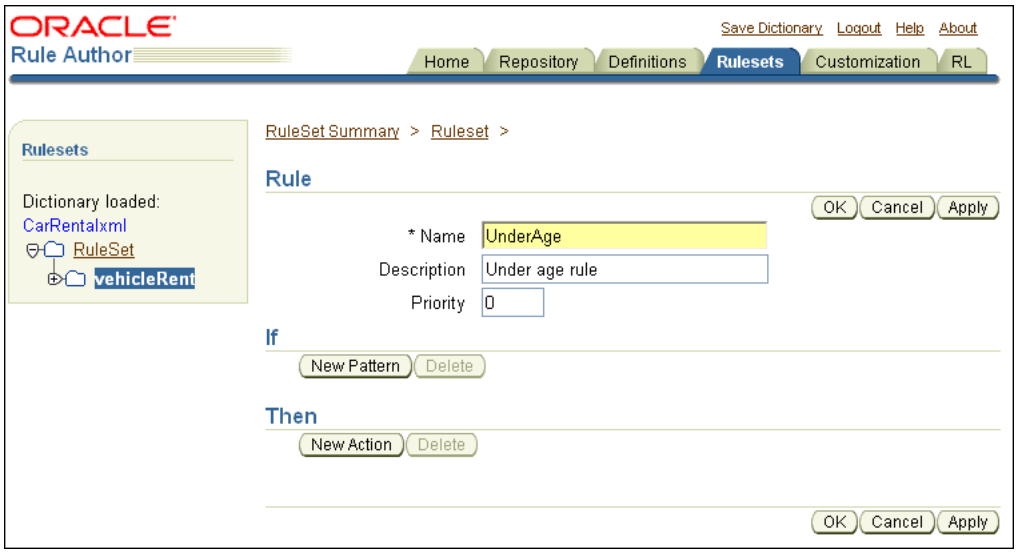


3. Click **Create**. This displays the Rule page.
4. On the Rule page, in the **Name** field, enter `UnderAge` .
5. On the Rule page, in the **Priority** field, enter `0`.

> **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 in a rule set are applied in any order until a decision is reached, and setting a priority is not required.

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

Figure 4–10 Rule Author Rule Page



4.6.2.2 Adding a Pattern to the UnderAge Rule (XML)

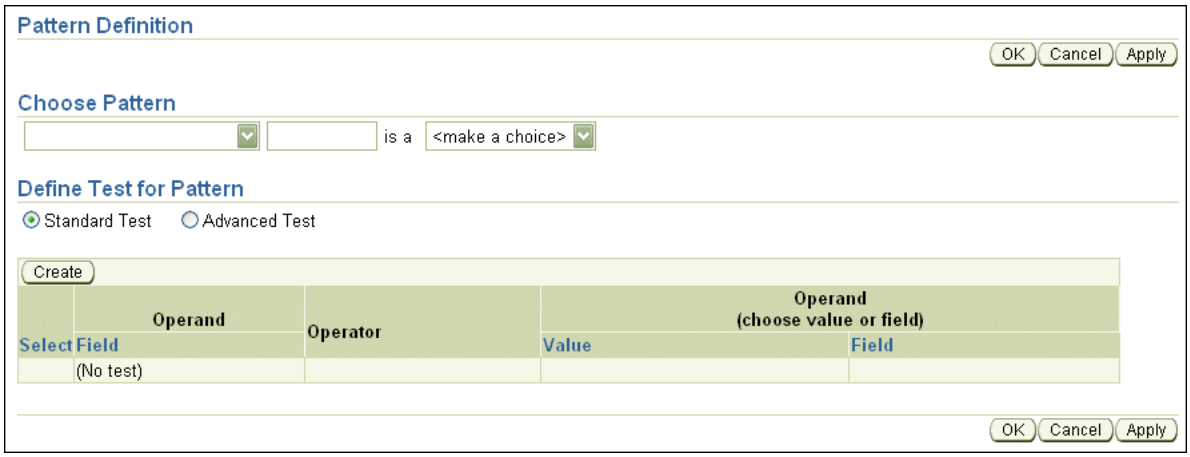
When Rules Engine runs, it uses the rules to check the available facts for matching patterns. To add a pattern for the UnderAge rule, do the following:

1. On the Rule page, click **New Pattern** in the **If** box. This displays the Pattern Definition page.

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

> **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 4–11 Rule Author Pattern Definition Page



2. 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).

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 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`.

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

5. Click **OK** to save the pattern definition and close the Pattern Definition page.

6. On the Rule page, click **OK** or **Apply** 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.

7. 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 4.6.2.3.

See Also: "Adding Actions for the UnderAge Rule (XML)" on page 4-20

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

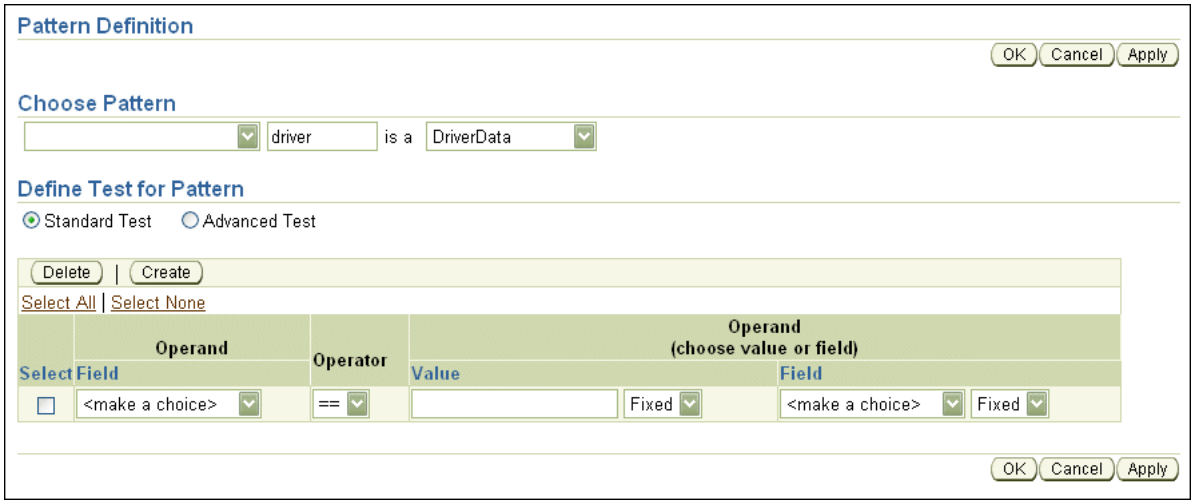
To add a test for a pattern, do the following:

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

2. In the **If** table on the rule page, select the pencil icon to display 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). For more information, see Section 3.8.1, "Using the Advanced Test Expression Option".

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

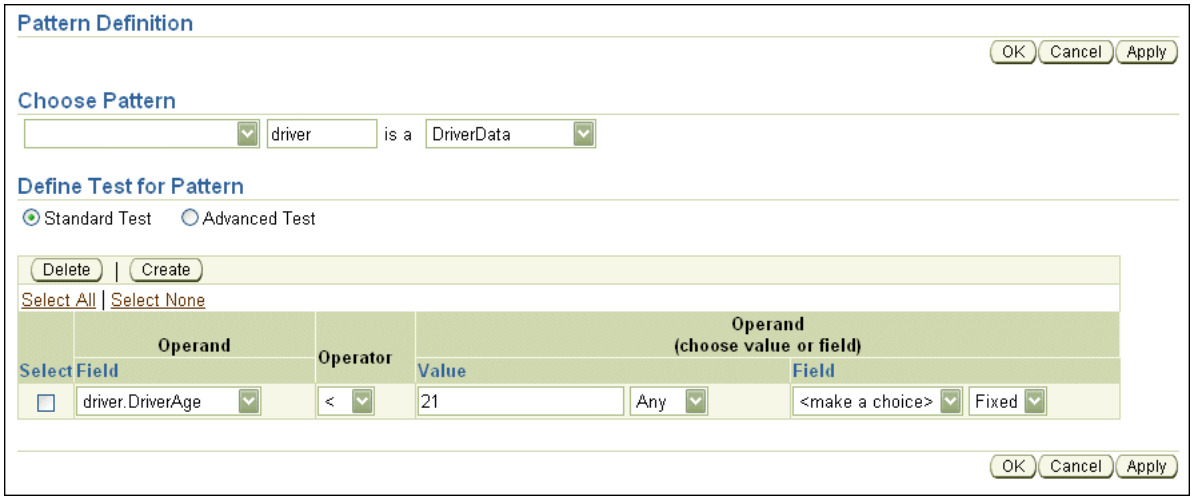


4. In the **Operand** column, from the **Field** box, select `driver.DriverAge`.
5. In the **Operator** column, select < (less than).
6. In the next **Operand** column, in the **Value** box enter 21. Do not enter a value in the **Field** box.
7. Next to each **Value** and **Field** box is a list containing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

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

   These values are called constraints. Use constraint values to enable or disable customization. Use the value `Fixed` to make the field read-only, which specifies that no customization is allowed for this value. 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 nontechnical users). You can also define constraints that allow you to limit the allowed values.

Figure 4–13 Rule Author Pattern Definition Page with Values for the UnderAge Rule



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.

Note the following when you define a test for a pattern:

- In **Standard Test** expressions, the tests only evaluate to true when all of the tests that you define match. Additionally, in standard tests, no grouping is allowed, and functions with parameters are not allowed. However, with standard tests you can define constraints for customization.
- In **Advanced Test** expressions, the tests do not have the restrictions of standard tests, but they do not allow the use of constraints. Advanced test expressions are not directly saved using RL Language because Rule Author incorporates aliases in the expressions; aliases are not supported in RL Language (Rule Author maps aliases to variable names).

> **See Also:** "Customizing Rules for the XML Car Rental Sample" on page 4-23

4.6.2.4 Adding Actions for the UnderAge Rule (XML)

Actions are associated with pattern matches. At runtime, when the "If" portion of a rule matches, Rules Engine executes the "Then" portion to run the action or actions associated with the rule.

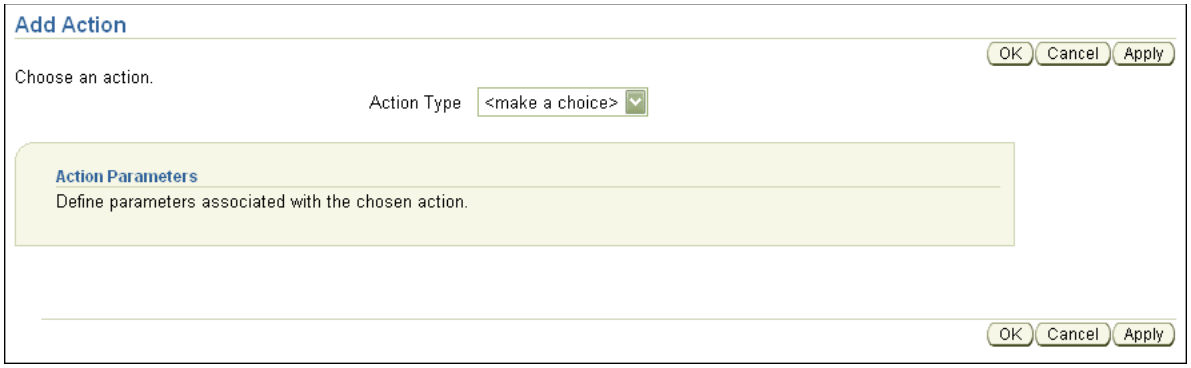
In this section, you add two actions for the UnderAge rule. The first action prints the result. The second action retracts the driver fact from the Rules Engine. You might want to retract a fact for a number of reasons, including:

- If you are done with the fact, and you want to remove it from Rules Engine.
- The action associated with the rule changes the state, so that the fact must be retracted to represent the current state of 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, under the vehicleRent folder, click the node for the UnderAge rule.
3. On the Rule page in the **Then** box, Click **New Action**. This displays the Add Action page (see Figure 4–14).

Figure 4–14 Rule Author Add Action Page



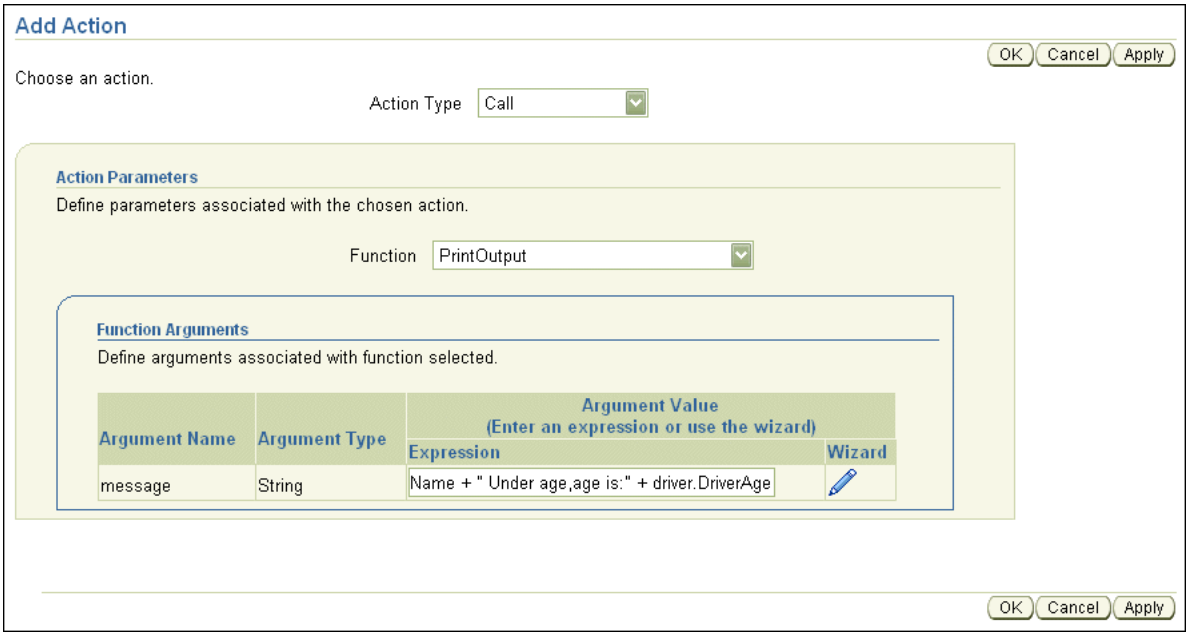
4. From the **Action Type** box, select the `Call` item. This shows the **Action Parameters** box.
5. From the **Function** box, choose `PrintOutput` (if you did not define an alias for `println`, then this is `DM.println`). This shows the **Function Arguments** box.
6. In the **Argument Value** field, enter the argument value (see Figure 4–15):

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

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

Note 2: You can also select the edit icon in the Wizard field to use the 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, because the Wizard presents a list showing the available variables.

Figure 4–15 Rule Author Add Action Page for the UnderAge Rule

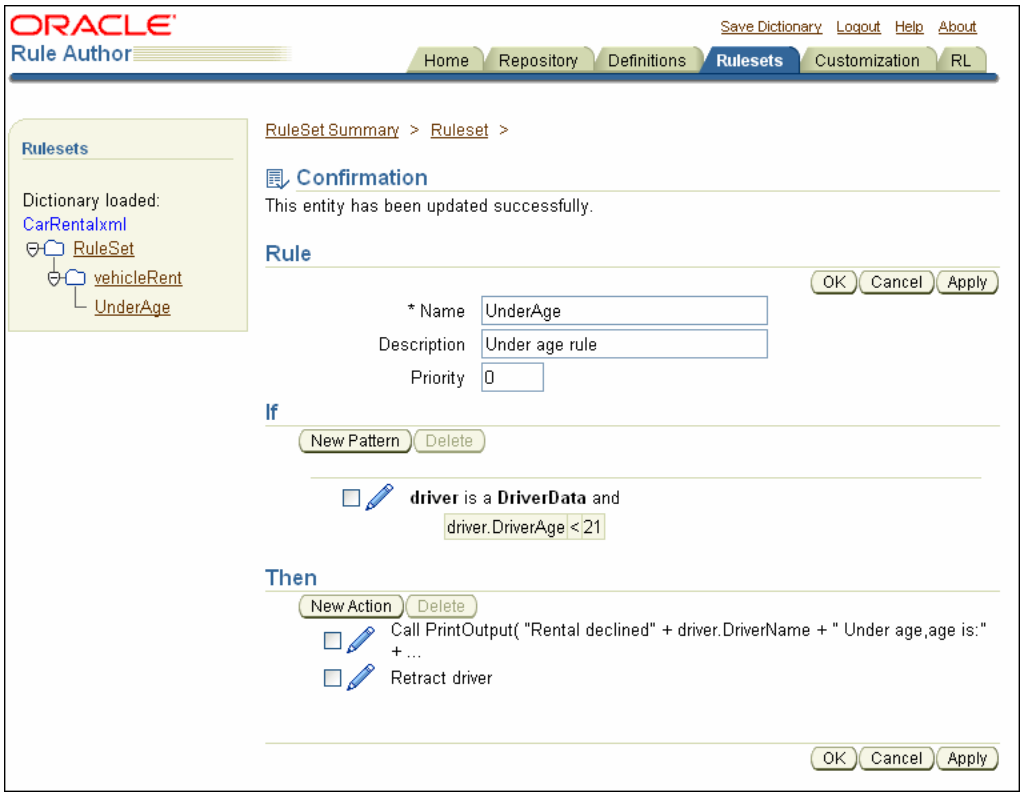


7. Click **OK** to save your changes and close the Add Action page.
8. On the Rule page, click **OK** or **Apply**.
9. Save the dictionary to save your work.

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. Under the vehicleRent folder, click the node for the UnderAge rule.
3. On the Rule page, from the **Then** box, click **New Action**. This displays the Add Action page.
4. From the **Action Type** box, select `Retract`. This shows the **Action Parameters** box.
5. From the **Fact Instance** box, select `driver`. The pattern name (`driver`) when used in the action, refers to a single instance that was matched by the pattern.
6. Click **OK** to save your changes and close the Add Action page.
7. On the Rule page, click **Ok** or **Apply** to save the changes (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: *Oracle Business Rules Language Reference Guide*

4.7 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 **Rulesets** 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 can 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 underage 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 node for the UnderAge rule followed by an asterisk (*), 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 the UnderAge 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 4-18

4.8 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 topics:

- Importing the Rules SDK and Rules RL Classes
- Creating a JAXB Context and Unmarshalling the XML Document
- Initialize the Repository with Rules SDK
- Loading A Dictionary with Rules SDK
- Loading a RuleSet and Generating RL Language for a 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 must create a JAXB context, and that you must use the `assertXPath` function to add facts to a rule session.

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.

4.8.1 Importing the Rules SDK and Rules RL Classes

The first step when you write a rule-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.sdk.store.jar.Keys;

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

4.8.2 Creating a JAXB Context and Unmarshalling the XML Document

Using the JAXB-generated classes, you first must specify a JAXB context and then 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));
```

4.8.3 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 (see Example 4–3):

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.
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 specifies the path to the repository, as shown with the `repoPath` parameter.
5. Use the `init` method in the `RuleRepository` object `repo` to initialize the repository instance.

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

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

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.

4.8.4 Loading A Dictionary with Rules SDK

When you build a rule-enabled Java application you must load a dictionary, with a specified version. Use a `RuleDictionary` object to load a dictionary, as shown in Example 4–4, 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 4–4 Loading a Dictionary With Rules SDK

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

4.8.5 Loading a RuleSet and Generating RL Language for a Data Model and Rule Set

After loading a dictionary, you can use Rules SDK to generate an RL Language program. This step is required, because a dictionary stores a data model and rule sets using an intermediate XML format. The `RuleDictionary` object 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 to all the rule sets in a dictionary, and the set of rules associated with a rule set.

Example 4–5 shows the code that generates the RL Language code for a rule set and for the associated data model.

Example 4–5 Generating Oracle Business Rules RL Language

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

4.8.6 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 Rules Engine and maintains the state of Rules Engine across a number of rule executions.

Example 4–6 shows the code that creates a `RuleSession` object and executes an RL Language program.

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

Note: The order of the `executeRuleset()` calls is important. You must 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 4–6 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, the rule session is ready to run the rule set against the facts that you assert for the rule session.

4.8.7 Asserting XML Data from Within a Rule Session

Before running a rule session, you first must unmarshall the XML document containing the XML data and then assert the facts from the XML document. Section 4.8.2 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 4–7 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 4–7 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 4-26

4.8.8 Using the Run Function with a Rule Session

Example 4–8 shows the code that runs a rule session.

Example 4–8 Running an Oracle Rules Engine Session

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

4.9 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 must 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 XML How-To.

Example 4–9 shows output from running `TestXML`.

Example 4–9 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 includes the following sections:

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 must 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 must register a rule execution set before running it. A registration associates a rule execution set with a URI; using the URI, you can create a JSR-94 rule session.

This section includes the following topics:

> **Note:** In Oracle Business Rules, a JSR-94 rule execution set registration is not persistent. Thus, you must 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 must 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 interdependencies are resolved. 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 must 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 interdependencies are resolved. 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 about JSR-94 extensions that assist you in including RL Language text

5.1.4 Creating a Rule Execution Set from RL Language 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 about 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 must list each `<rule-source>` in the order in which they are to be interpreted in 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 includes 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` object.

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` exception.

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 Language 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. 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 Language 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 rule-enabled applications.

You can use the Rules SDK APIs in a rule-enabled application to access existing rules and then run 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 includes the following sections:

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 JavaBeans 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 the HashMap style is necessary for at least one GUI framework.

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` object, you can use the Rules SDK to create a data model in the dictionary. A `RuleDictionary` object 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 e-mail address	Java FactType
`email.EmailAddressList`	Represents a list of e-mail 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` object 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 e-mail message is spam. Table 6–4 shows these data model components.

Table 6–4 Sample Data Model Types for Handling the E-mail Package

Name	Description	Type
`SpamFound`	Asserted when an e-mail 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 Data Model

After you create and open a `RuleDictionary` object, you can use Rules SDK to create an `editor.DataModel` instance. The `RuleDictionary` object 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` object. A `FactType` object 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`). 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 JavaBeans defined properties, and methods.

6.3.2 Creating Data Model Components

You create each part of the data model using the appropriate `ModelComponentTable` object. 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

In Rules SDK you create argument lists for functions using a `FormalParameterTable` object. 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` object has a type and a name. The type of a FormalParameter is selected from the available FactTypes.

The code in Example 6–5 creates a `FormalParameterTable` object 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

In 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

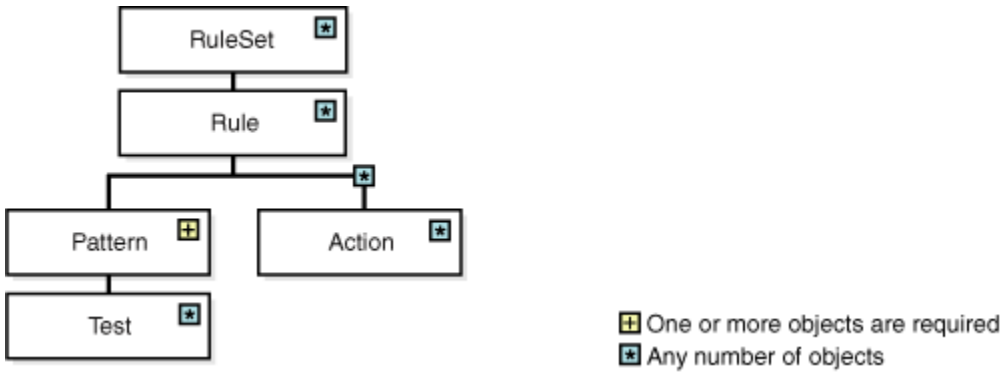
```
//set the body of the function
// in this case just pretty print a message
fKillSpam.setBody(" println(\" email from: \" + emsg.getSender() + \"  because \"
+ reason)" );
```

6.4 Using Rule Sets and Creating and Modifying Rules

In 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 patterns (order has meaning). Rule actions can be calls, assignments, retractions, and assertions.

Figure 6–1 shows the general container hierarchy for a RuleSet.

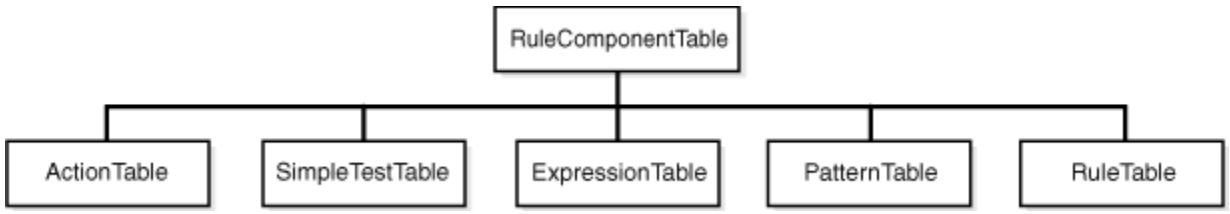
Figure 6–1 Rules SDK RuleSet Container Hierarchy



Rules SDK provides classes that represent each of these objects; all classes descend from `RuleComponent`. For collections, for example rules in a rule set, patterns in a rule, or actions in a rule, Rules SDK provides a class that is descended from the

`RuleComponentTable` to manage a specific collection (see Figure 6–2). The `RuleComponentTable` subclass provides a specialized `add()` method for the particular subclass 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 Rule Set

Generally, the sequence you use to create a `RuleComponent` object is:

1. Create a `RuleSet` instance by use of a `RuleSet` constructor.
2. 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` subclass provides a specialized `add()` method for the particular subclass represented by the table.

The code in Example 6–8 creates a `RuleSet` for the e-mail 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 Rule Set

Rules SDK provides classes that represent the objects in a `RuleSet`. Each of these objects descend from `RuleComponent` object. For collections, for example rules in a ruleset, or patterns in a rule or actions in a rule, Rules SDK provides classes that are descended from `RuleComponentTable` class to manage a specific collection. The

`RuleComponentTable` subclass provides a specialized `add()` method for the particular subclass represented by the table.

Several classes are not parts of collections. To access these classes, use 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 Rule Set.

Example 6–9 Adding a Rule to a Table in a Rule Set

```
//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` subclass 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` objects or `AdvancedExpression` objects A `Pattern` may have an unlimited number of tests.

The Rules SDK "ANDs" tests together when generating RL Language. For example, the test used for the e-mail example requires a function with parameters. This requires an `AdvancedExpression` object (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.

This appendix includes the following sections:

A.1 Rule Author Naming Conventions

This section covers Rule Author naming conventions.

A.1.1 Rule Set Naming

Rule Author enforces a limitation for rule set names; a rule set name can contain only the characters (a to z and A to Z) and numbers (0 to 9), or an underscore character (_).

A.1.2 Dictionary Naming

Rule Author dictionary names can contain only the following characters, upper and lowercase letters (`a` to `z` and `A` to `Z`), numbers (`0` to `9`), the period (`.`) the underscore character (_), 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 create another dictionary named `TEST` only if you first delete the dictionary named `Test`.

Additionally, dictionary names must contain at least one letter. For example, the dictionary named `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 contain only the characters (`a` to `z` and `A` to `Z`), numbers (`0` to `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 create another version named `TEST` only if you first delete the version named `Test`.

A.1.4 Alias Naming

A Rule Author alias names can contain any characters, including a single space. When you use an alias in an expression, if the alias begins with a letter, a dollar sign ($), or an underscore (_) and contains only these characters, it does not have to be enclosed in quotation marks.

When you use an alias containing special characters or embedded spaces in an advanced expression, the alias must be enclosed 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 because Rule Author sessions timeout after a period of inactivity. You specify the timeout period in the Rule Author application `web.xml` file using the `<session-timeout>` element.

When Rule Author times out, it automatically saves the current work for the loaded dictionary to a version named `SCRATCH_`<*login user name*> in the repository, where *login user name* is the name of the logged in Rule Author user.

Rule Author only saves one `SCRATCH_` file per user. Thus, if Rule Author times out once, the `SCTATCH_` version is saved to the repository with the log in user name. When the same user logs in again, and connects to the same repository, Rule Author shows a warning message indicating that the `SCRATCH` version should be saved to a new version name. If Rule Author times out again, and you do not save the `SCRATCH` version to a different version, the second and any subsequent timeouts overwrite any existing `SCRATCH` version (on a per Rule Author user basis).

When Rule Author times out, you are redirected to the Oracle Single Sign-On login page. When Oracle Single Sign-On is not enabled, after a timeout you are redirected to the Rule Author default authentication page.

> **See Also:** Section C.9, "How Do I Modify the Single Sign-On Timeout for Oracle Application Server?"

A.3 Rules SDK and Rule Author Temporary Files

When you update a file repository, the Rules SDK creates and 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. However, it is possible that due to certain abnormal termination conditions, these temporary files can 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 about using Rule Author and Rules SDK with repositories.

This appendix includes the following sections:

B.1 Working with a WebDAV Repository

This section contains information about setting up and configuring a WebDAV repository.

B.1.1 Setting up a WebDAV Repository

Oracle Business Rules supports using a WebDAV repository as the persistent storage for rule sets, the data model, and rules. This section describes how to set up a WebDAV repository and presents basic instructions for setting up a file system based WebDAV repository in Oracle HTTP Server. Oracle HTTP Server supports WebDAV with the `mod_oradav` module.

The WebDAV protocol is an extension to the HTTP protocol that enables remote users to write content to the Web server. Using this configuration, it is important that the Web server is properly configured to prevent undesirable consequences and to ensure the that a secure system is maintained.

It is strongly recommended that you employ some or all of the following security features on the Web server:

- 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 you can use to establish a WebDAV based repository 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:`port*.

> **Note:** Only use this example configuration for the WebDAV repository for internal testing and not for an actual production environment. This configuration does not include access control, and therefore allows anyone to access or modify the WebDAV repository. Please refer to Section B.2 for information about configuring a WebDAV repository with 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.

These instructions establish a WebDAV repository accessible with the following URL:

`http://www.fully_qualified_host_name.com:port/rule_repository/`

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

See Also: *Oracle HTTP Server Administrator's Guide* for information about configuring and using `mod_oradav`. In particular, see the section titled "WebDAV Security Considerations" in Chapter 9

B.1.2 Connecting to a WebDAV Repository

When you select the WebDAV repository type 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.
`User Name`	Specifies the user authorized for WebDAV access.
`Password`	Specifies the password for the WebDAV user associated with the specified User Name.

Note: In Rule Author when you supply both the user name and password and other required properties, and also specify an Oracle Wallet, the properties that you specify in the dialog take precedence over the Oracle Wallet information.

B.1.3 Connecting to a WebDav Repository Using a Proxy

Rule Author looks for the `http.proxyHost` system property. If this property is set, then the Rule Author picks up the http proxy system properties and uses them for the WebDAV connection. There are three properties you can set to specify that the http protocol handler uses a proxy:

- `http.proxyHost`: the host name of the proxy server
- `http.proxyPort`: the port number, the default value being 80
- `http.nonProxyHosts`: a list of hosts that should be reached directly, bypassing the proxy. This is a list of regular expressions separated by '|'. Any host matching one of these regular expressions will be reached through a direct connection instead of through a proxy.

When a proxy is required to access the WebDAV repository, Rule Author displays the parameters shown in Table B–2, as well as those shown in Table B–1.

Table B–2 Configuration Parameters for Connecting to WebDAV Repository with Proxy

Parameter	Description
`Proxy User Name`	Specifies the proxy user name. This is required if the proxy server is configured with security.

Table B–2 (Cont.) Configuration Parameters for Connecting to WebDAV Repository with

Parameter	Description
`Proxy Password`	Specifies the proxy password. This is required if the proxy server is configured with security.

B.2 WebDAV Repository Security

WebDAV allows read and write access to a WebDAV enabled Web server. It is highly recommended that you take the appropriate steps to secure the WebDAV Web server. To this end, you should encrypt, using SSL, connections to a WebDAV Web Server and you should also require authentication.

This section covers the following topics:

- Communicating with a WebDAV Repository Over SSL from Rule Author
- Setting the Location of Your Oracle Wallet
- Configuring Rule Author for WebDAV Repository Authentication
- Storing Data in an Oracle Wallet for WebDAV Repository Authentication

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, 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.

You can also create a new file repository by clicking **Create** on the repository connect page, when the Repository type selected is File. If you enter an existing repository path

and click **Create**, the create behaves as if you clicked **Connect**, and connects you to the existing 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 repository, you should add the repository to the OracleAS Recovery Manager configuration so that the repository is included in the backup and recovery process.

For more information about OracleAS Recovery Manager, see Oracle Application Server Administrator's Guide.

> **See Also:** Oracle Application Server High Availability Guide for information on Oracle Business Rules and high availability

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:

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.

> **Note:** 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.1.4 How Do I Use Rules SDK to Include a null in an Expression

The Rules SDK automatically adds quotes for String values. To include a value without quotes, for example a null value without quotes in an expression, specify an advanced expression. The expression type `FORM_LITERAL` always surrounds string values with quotes. When you use the expression type `FORM_ADVANCED`, this does not surround the expression with quotes.

For example, the following Rules SDK code includes a null:

```
SimpleTest test = pattern.getSimpleTestTable().add();
test.getLeft().setForm(Expression.FORM_SINGLE_TERM);
test.getLeft().setSingleTermValue(attr);
test.setOperator(Util.TESTOP_NE);
test.getRight().setForm(Expression.FORM_ADVANCED);
test.getRight().getAdvancedExpression().append("null");
```

If you did not use the `FORM_ADVANCED`, you might see the following errors:

```
SEVERE: RUL-01815: Null values not allowed for key or value put ( LiteralValue,
null )

java.lang.NullPointerException: RUL-01815: Null values not allowed for key or
value put ( LiteralValue, null )
```

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 a WebDAV repository.

C.3 How do I Deploy Rule Author on Non-Oracle Containers?

This section includes instructions for deploying Rule Author and the online help for Rule Author on non-Oracle containers.

This section covers the following:

- Deploying Rule Author on WebSphere V6.1 (WAS V6.1)
- Deploying Rule Author on WebLogic Server
- Deploying Rule Author on JBoss 4.0

C.3.1 Deploying Rule Author on WebSphere V6.1 (WAS V6.1)

To deploy Rule Author on WebSphere V6.1, first locate the OracleAS Companion CD installation disk supplied with Oracle Application Server. The OracleAS Companion CD Disk 2 includes two .ear files for each supported non-Oracle container. The .ear files for WebSphere should be located in the directory:

```
Disk2/rules/webapps/websphere/
```

Note: Under WebSphere, Rule Author online help is not available.

It is recommended that your installation of WebSphere have administrative security configured. Please consult the WebSphere documentation on setting up administrative security.

As part of the installation of Rule Author, you must configure the appropriate user and group name for authentication and authorization. The following steps provide only an example of how to setup the user and password for WebSphere WAS V6.1 using the standalone custom registry. Production uses of Rule Author should have security configured by an expert. To see examples of more advanced approaches, such as database or LDAP, consult the appropriate IBM documentation.

Perform the following steps to deploy Oracle Business Rules Rule Author on WebSphere V6.1:

1. Start the WebSphere server:

```
% cd <websphere home>/profiles/AppSvr01/bin
% ./startServer.sh server1
```

2. Login to WebSphere administrative console using the URL,

```
https://<host name>:9043/ibm/console/logon.jsp
```

Note: If you enable global security, you have to log in to the admin console with the user name that belongs to the admin group.

---

   a. Click to expand the "Security" node on the navigation tree.

   b. Click the "Secure administration, applications, and infrastructure" node.

   c. In the "User account repository" group box, select "Standalone custom registry" from "Available realm definitions" and click "Set as current".

   d. Click **Apply**.

   e. Click the Save link in the "messages" box at the top of the page.

   f. On the server's file system, choose a directory that is only accessible by the operating system user that WebSphere is running as, and create a group file and user file within that directory. This creates two users, adminUser who belongs to the AdminGroup group with special privilege to log in to the admin console, and ruleUser1 who will be given permission to access the Rule Author application.

      The following shows an example of the contents of the group file groups.prop

```
adminGroup:987:adminUser:AdminGroup
ruleAdminGroup:567:ruleUser1:RuleAdminGroup
```

      Each line of the group file should contain the following:

```
<group name>:<group id>:<user name 1>, .. , <user name N>:<display name>
```

      The following shows sample contents of user file users.prop:

```
adminUser:welcome1:123:987:WebSphereAdmin
ruleUser1:welcome1:456:567:RuleAdmin
```

      Each line of user file should contain the following:

```
<name>:<password>:<unique user id>:<group name 1>,...,<group name
N>:<display name>
```

      This sets the passwords for both adminUser and ruleUser1 to welcome1.

g. In the "User account repository" group box, select "Standalone custom registry" from "Available realm definitions" and click "Configure".

h. Click the "Custom properties" link to set up two properties that specifies the location of the files:

```
Name Value
---------------------------
groupsFile <your path>/<group file name>
usersFile <your path>/<user file name>
```

i. When you finish configuring these 2 properties, click the "Standalone custom registry" link at the top to return to the previous page.

j. Set the "Primary administrative user name" to "adminUser".

k. Select the "Server identity that is stored in the repository" radio button.

l. Type in the Server user ID ("adminUser") and server password ("welcome1") that were previously defined in the users.prop file into fields of the same name. Note that the user name and server password must be included in the user file and group file that you provided earlier (see Step d).

m. Enter "com.ibm.websphere.security.FileRegistrySample" into "Custom registry class name" field (this should be the default value).

n. Click "OK".

o. Check the "Enable application security" box.

p. Uncheck the "Use Java 2 security" box if it is checked. Under some circumstances, this box is checked automatically when checking the box in the previous step and must be unchecked.

q. Click the "Apply" button.

r. Click "Save" in the "messages" box at the top of the page.

3. Click to expand the "Servers" node in the left column, then click "Application servers". Click the server you wish to deploy the Rule Author to.

4. Stop and restart the server from the command line with the `stopServer` and `startServer` commands.

5. Click Applications->Install New Application Node on the left navigation panel. You have to specify the path to the ruleauthor_websphere.ear and click "Next".

   Change the field "ApplicationName" from "Rule Author" (with a space) to "RuleAuthor" (with no space). Click "Next", and then "Next" for the next two screens. Finally, click "Finish".

   - Click on Environment in the left panel.
   - Click on Shared Libraries
   - Select the desired scope from the Scope menu.
   - Click "new"
   - Enter "Oracle XDK" for the name
   - Under Classpath, enter the absolute path to the files "xml.jar" and "xmlparserv2.jar", which are in the location of the deployed RuleAuthor.ear application. (find using find or search). Enter the paths one per line with no separator (for example, ":") between them.
   - Click the "OK" button

- Click "Save"

6. In the "Applications" area, click on "Enterpirse Applications", then click the RuleAuthor link.
   - Under the References heading, click on "Shared library references".
   - Select the checkbox for the "RuleAuthor" entry and click the "Reference shared libraries" button.
   - Select the "Oracle XDK" entry in the "Available" list and move it to the "Selected" list.
   - Click "OK".
   - Click "OK" on the next page.
7. Click "Security role to user/group mapping".
8. On the "Security role to user/group mapping" screen, check the checkbox in the "Select" column and then click the "Look up groups" button.
9. Click the "Search" button to display the available groups, then move the user "group1" from the "Available" box to the "Selected" box. Click "OK".
10. On the returned page, click "Save".
11. Logout of the WebSphere console.
12. Stop and then restart the WebSphere instance from the command line.
13. Point your browser to the following URL (the default port is 9080):

    ```
    http://<host name>:<port>/ruleauthor
    ```

If you have mapped the RuleAdministrator role in Step 2, you see the Rule Author login page when you attempt to click on any link on the page. If you have not correctly mapped the RuleAdministrator role, the application will go directly to the Repository Connection page.

Password-protected WebDAV Repository access using Oracle Wallet is currently non-functional on WebSphere. Password-protected access to a WebDAV repository only works using a directly-specified username and password.

Note: To import an XML schema, first add the files xml.jar and xmlparserv2.jar (located in the installed Rule Author application directory) to the Java Fact classpath. The Java class files generated by the JAXB compiler from the XML schema depend on several classes provided by these JAR files. In non-Oracle containers, these classes are not on the Java Fact Classpath by default, so they must be added manually.

C.3.2 Deploying Rule Author on WebLogic Server

To deploy Oracle Business Rules Rule Author and the online help for Rule Author on WebLogic Server 9.1, you first need to locate the OracleAS Companion CD installation disk supplied with Oracle Application Server. The OracleAS Companion CD Disk 2 includes two .ear files for each supported non-Oracle container.

Perform the following steps to use the exploded ear file approach to deploy Oracle Business Rules Rule Author on WebLogic Server 9.1.

Note: Locate the weblogic .ear files, `ruleauthor_weblogic.ear` and `rulehelp_weblogic.ear` on the OracleAS Companion CD Disk 2, in the directory,

`Disk2/rules/webapps/weblogic/`

1. Make a directory named `ruleauthor` somewhere in <*bea home*>, where *bea home* is the WebLogic Server 9.1 home directory.
2. Go to this directory and expand `ruleauthor_weblogic.ear`, located on the OracleAS Companion CD Disk 2, in the directory, as follows:

   ```
   % cd ruleauthor
   % jar xvf path/Disk2/rules/webapps/weblogic/ruleauthor_weblogic.ear
   ```

3. Start WebLogic Server, as follows:

   ```
   % cd <bea home>/weblogic91/samples/domains/wl_server/bin
   % startWebLogic.cmd (for Windows)
   % ./startWebLogic.sh (for Unix)
   ```

4. Point your browser to WebLogic Server Administration Console and log in. For example

   `http://<hostname>:7001/console/login/LoginForm.jsp`

5. On the left panel, click "Deployments".
6. On the left panel, click the "Lock & Edit" button.
7. Click the "Install" button.
8. Navigate the file browser until your find `webapp` directory. Select the radio button for this directory. Follow the instruction to deploy the application.
9. On the left panel, click the "Activate Changes" button.
10. On the left panel, click the "Lock & Edit" button.
11. On the left panel, click "Security Realms".
12. Click "myrealm".
13. Click the "Users and Groups" tab.
14. Click the "Groups" subtab
15. Click the "New" button. Create a new group "rule-administrators".
16. Click the "Users" subtab.
17. Click the "New" button. Create a new user. Enter your choice of name. For example, enter `ruleadmin` and enter a password of the new user and then click "Save". Click the "Groups" tab. Using the shuttle, assign `rule-administrators` group to this user. Click "Save" when complete.
18. Click the "Release Configuration" button.
19. Go to the "Deployments" page and start the application.
20. Log out of the console and close the browser.
21. To test the application, open the browser and go to the URL

    `http://<localhost>:7001/ruleauthor/`

To deploy the help for Rule Author, follow Steps (1-7) to deploy `rulehelp_weblogic.ear`. To test the help page, click the help link on the Rule Author page.

> **Note:** To enable XML Fact Import and the use of WebDAV repository access, add the Oracle XDK classes to the container classpath. To do this, update `startWebLogic.sh` for the domain you are running by adding `xml.jar` and `xmlparserv2.jar` to the `CLASSPATH` environment variable path (on Windows, the name of the file you need to update is `startWebLogic.cmd`).

C.3.3 Deploying Rule Author on JBoss 4.0

To deploy Rule Author and the online help for Rule Author on JBoss, you first need to locate the OracleAS Companion CD installation disk supplied with Oracle Application Server. The OracleAS Companion CD Disk 2 includes two .ear files for each supported non-Oracle container.

> **Note 1:** If you wish to use the Rule Author XML Schema Import capability, you must add the locations of the files xml.jar and xmlparserv2.jar to the environment variable JBOSS_CLASSPATH in the environment from which you execute JBoss. These classes are available in ORACLE_HOME/lib.

> **Note 2:** Under JBoss, you cannot access WebDAV repositories due to XML parser incompatibilities.

Perform the following steps to deploy Rule Author on JBoss 4.0.

> **Note:** Locate the JBoss .ear files, `ruleauthor_jboss.ear` and `rulehelp_jboss.ear` on the OracleAS Companion CD Disk 2, in the directory,
>
> `Disk2/rules/webapps/jboss/`

1. Make sure you have JDK 1.5 installed in your environment. JBoss does not run well with JDK 1.4.
2. Put `ruleauthor_jboss.ear` in the <*jboss home*>/server/default/deploy directory.
3. Put the following jars in the <*jboss home*>/server/default/lib directory. They can be obtained by expanding the `ruleauthor_jboss.ear` file in a temporary directory.

```
commons-el.jar
jr_dav.jar
jsp-el-api.jar
oracle-el.jar
regexp.jar
rulesmvc.jar
rulesdk.jar
rl.jar
share.jar
```

```
uix2.jar
webdavrc.jar
xmlparserv2.jar
xml.jar
http_client.jar
oraclepki.jar
servlet.jar
```

4. Edit <*jboss home*>/server/default/conf/login-config.xml to add the following section:

```
<application-policy name = "ruleauthor">
   <authentication>
      <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
          flag = "required">
         <module-option name= "usersProperties">
                  props/ruleauthor-users.properties
         </module-option>
         <module-option name="rolesProperties">
                  props/ruleauthor-roles.properties
         </module-option>
      </login-module>
   </authentication>
</application-policy>
```

5. Add the users (`ruleauthor-users.properties`) and roles (`ruleauthor-roles.properties`) files to the following directory:

   <*jboss home*>/server/default/conf/props

   Example of contents of `ruleauthor-users.properties` file:

   ```
   ra=oraclerule
   ```

   Example of contents of `ruleauthor-roles.properties` file:

   ```
   ra=RuleAdministrator
   ```

   In this example, we create a user name `ra` that has a password `oraclerule` and belongs to role `RuleAdministrator`.

6. Start JBoss using the following commands,

   ```
   %cd <jboss home>/bin
   %run.bat
   ```

   If successfully deployed, you should see output similar to the following:

   ```
   14:28:21,718 INFO [TomcatDeployer] deploy,ctxPath=/ruleauthor, warUrl=.../tmp/
   deploy/tmp20477ruleauthor_jboss.ear-contents/ruleauthor-exp.war/

   4:28:21,906 INFO  [EARDeployer] Started J2EE application:
   file:/C:/jboss-4.0.3SP1/server/default/deploy/ruleauthor_jboss.ear
   ```

7. Point your browser to the site,

   http://<host>:8080/ruleauthor

   If you have mapped the `RuleAdministator` role as instructed in Steps 2 and 3, you see the Rule Author login page.

To deploy the help for Rule Author, drop the `rulehelp_jboss.ear` to the deploy directory. To test the help page, click the help link on the Rule Author page.

C.4 How Does a RuleSession Handle Concurrency and Synchronization?

Method calls on an Oracle Business Rules RuleSession object are thread-safe such that calls by multiple threads do not cause exceptions at the RuleSession level. However, there are no exclusivity or transactional guarantees on the execution of methods. The lowest-level `run` method in the Rules Engine is synchronized, so two threads with a shared `RuleSession` cannot both simultaneously execute `run`. One call to run must wait for the other to finish.

RL Functions are not synchronized by default. Like Java methods, RL functions can execute concurrently and it is the programmer's responsibility to use synchronized blocks to protect access to shared data (for instance, a `HashMap` containing results data).

Any set of actions that a user wishes to be executed as in a transaction-like form must synchronize around the shared object. Users should not synchronize around a `RuleSession` object because exceptions thrown when calling `RuleSession` methods may require the `RuleSession` object to be discarded.

For most uses of a `RuleSession` object in Oracle Business Rules, each thread or servlet instance should create and use a local `RuleSession` object. This usage pattern is roughly analogous to using a JDBC connection in this manner.

The following examples demonstrate how to use a shared `RuleSession` object.

For the case where Thread-1 includes the following:

```
ruleSession.callFunctionWithArgument("assert", singleFact1);
ruleSession.callFunctionWithArgument("assert", singleFact2);
```

and Thread-2 includes the following:

```
ruleSession.callFunction("run");
ruleSession.callFunction("clear");
```

In this case, the execution of the two threads might proceed as shown in Example C–1.

Example C–1 Using a Shared RuleSession Object in Oracle Business Rules

```
Thread-1:  ruleSession.callFunctionWithArgument("assert", singleFact1);
Thread-2:  ruleSession.callFunction("run");
Thread-2:  ruleSession.callFunction("clear");
Thread-1:  ruleSession.callFunctionWithArgument("assert", singleFact2);
```

In Example C–1, the two facts Thread-1 asserted are never both in the `RuleSession` during a call to `run`. Notice also that only one thread calls the `run` method. We do not recommend a design where multiple threads can call `run` on a shared `RuleSession`. This usage pattern can create extremely hard to find bugs and there is no gain in performance.

All accesses to a shared `RuleSession` object must be synchronized to ensure the intended behavior. However, a `RuleSession` instance may throw an exception and not be recoverable, so do not use this object as the synchronization object. Instead, use another shared object as the synchronization point.

One can envision a shared server process producer-consumer model for `RuleSession` use. In this model, multiple threads assert facts to a shared `RuleSession` and one thread periodically calls `run`, reads any results, and outputs them. This ensures that thread conflicts cannot occur, because the two code segments must be executed serially and cannot be intermingled. For example, the code with

shared objects, producer code, and consumer code in Example C–2, Example C–3, and Example C–4.

Example C–2 RuleSession Shared Objects

```
RuleSession ruleSession;
Object ruleSessionLock = new Object();
```

Example C–3 RuleSession Producer Code

```
public String addFacts(FactTypeA fa, FactTypeB fb, FactTypeC fc){
   String status = "";
   synchronized(ruleSessionLock){
     try {
       ruleSession.callFunctionWithArgument("assert", fa);
       ruleSession.callFunctionWithArgument("assert", fb);
       status = "success";
     } catch (Exception e) {
       // a method that creates a new RuleSession loads it with rules
       initializeRuleSession();
       status = "failure";
     }
     return status;
}
```

Example C–4 RuleSession Consumer Code

```
public List exec(){
  synchronized(ruleSessionLock){
    try {
      ruleSession.callFunction("run");
      List results = (List)ruleSession.callFunction("getResults");
      ruleSession.callFunction("clearResults");
      return results;
    }  catch (Exception e) {
      // a method that creates a new RuleSession loads it with rules
      initializeRuleSession();
      return null;
    }
  }
}
```

Note: When multiple threads are sharing a `RuleSession` object, it is not recommended that more than one of the threads calls the `run` method.

C.5 How Do I Improve Oracle Business Rules Runtime Performance?

When trying to increase the performance of an application that uses a `RuleSession` object, you should try to minimize the number of times that you open the Repository and generate the RL code. Thus, you should avoid opening and reading a Repository for each `RuleSession` instance you create; this practice should allow the program to perform well in most cases.

You can share the `String` object containing generated RL code, so that all application instances can use it for loading the rules into a private `RuleSession` object.

If your application requires additional performance tuning, you can use a pooling mechanism to share a set of `RuleSession` objects. In this case, a common pool would hold `RuleSession` objects and a thread could obtain an object from the pool, use the object for a series of operations, and then return the object to the pool. `RuleSession` objects which throw an exception should not be reused. If an exception was thrown by the `RuleSession`, the pool would need to discard this instance and replace it with a new `RuleSession` instance.

Note the following issues when using pooled `RuleSession` objects:

- If the rule actions do not perform input or output, including calls to external resources, the size of the pool does not need to be larger than the number of CPUs in the machine.
- The `RuleSession` state must be restored prior to reuse. Use of the RL `reset()` function may be helpful. Pay particular attention to the fact that `reset()` re-executes initializers for non-final global variables in the `RuleSession`. This can be used as a hook to re-assert initial facts.
- Do not reuse a `RuleSession` that throws an exception, because it is difficult to be sure that the state is restored for `RuleSession` instances that throw an exception.

Depending on the computation requirements of your application, the performance improvements from pooling may be very small. The computational cost of creating a new `RuleSession` object and loading RL code is unlikely to be a performance bottleneck. You should monitor performance to see if it is within the acceptable range before attempting to increase performance using `RuleSession` object pooling.

The Oracle Java Object Cache provides a mechanism for very simple pooling. However, you will need to add an extra layer that can call `reset()` and check for exceptions that require `RuleSession` objects to be discarded and re-created.

> **See Also:** Java Object Cache in the *Oracle Containers for J2EE Services Guide*

C.6 How Do I Correctly Use an RL Language Cross Product?

When working with cross products of facts, there are cases where the runtime behavior of RL Language may produce surprising results.

Consider the RL Language code in Example C–5.

Example C–5 Cross Product Using Fact F

```
class F {int i; };
rule r1 {
  if (fact F F1 && fact F F2) {
    println("Results: " + F1.i + ", " + F2.i);
}}
assert(new F(i:1));
assert(new F(i:2));
run();
```

How many lines print in the Example C–5 output? The answer is 4 lines because fact `F1` and fact `F2` can be the same instance and are compared in each operand.

Thus, Example C–5 gives the following output:

```
Results: 2, 2
Results: 2, 1
Results: 1, 2
Results: 1, 1
```

Using the same example with a third `F`, for example (`assert(new F(i:3));`) then nine lines are printed and if, at the same time, a third term `&& fact F F3` is added then 27 lines are printed.

If you are attempting to find all combinations and orders of distinct facts, you need an additional term to in the test, as shown in Example C–6.

Example C–6 Find All Combinations of Fact F

```
rule r1 {
  if (fact F F1 && fact F F2 && F1 != F2) {
    println("Results: " + F1.i + ", " + F2.i);
}}
```

The code in Example C–6 gives the following output:

```
Results: 2, 1
Results: 1, 2
```

The simplest, although not the fastest way to find all combinations of facts, regardless of their order, is to use the code shown in Example C–7.

Example C–7 Finding Combinations of Fact F

```
rule r1 {
  if (fact F F1 && fact F F2 && id(F1) < id(F2)) {
    println("Results: " + F1.i + ", " + F2.i);
  }
}
```

Because the function `id()` shown in Example C–7 takes longer to execute in a test pattern than a direct comparison, the fastest method is to test on a unique value in each object. For example, you could add an integer value property "oid" to your class that is assigned a unique value for each instance of the class.

Example C–8 shows the same rule using the oid value.

Example C–8 Fast Complete Comparison

```
rule r1 {
  if (fact F F1 && fact F F2 && F1.oid < F2.oid) {
    println("Results: " + F1.i + ", " + F2.i);
  }
}
```

This problem may also arise if you attempt to remove all duplicate facts from the Oracle Rules Engine, using a function as shown Example C–9.

Example C–9 Retracting Duplicate Facts Incorrect Sample

```
rule rRemoveDups {
  if (fact F F1 && fact F F2 && F1.i == F2.i) {
     retract(F2);
```

```
  }
}
```

However, this rule removes all facts of type `F`, not just the duplicates because `F1` and `F2` may be the same fact instance. Example C–10 shows the correct version of this rule.

Example C–10 Retracting Duplicate Facts Corrected Sample

```
rule rRemoveDups {
  if (fact F F1 && fact F F2 && F1 != F2 && F1.i == F2.i) {
     retract(F2);
  }
}
```

C.7 How Do I Access a Rule in a Dictionary Using a URL?

Rule Author can take a dictionary and a rule name in a URL and access a rule directly from the URL. After a user logs in to Rule Author either through the Rule Author login page using Oracle Single Sign-On, Rule Author allows the user to display a ruleset or rule page with a URL.

The format of the URL is as follows, with parameter descriptions in Table C–2.

```
http://<host_name>:<port_number>/ruleauthor/event=dlConnect&
<repos_prop 1>=<repos_value val> ..... &
Dict=<dict_name>&Version=<version> &
RuleSet=<ruleset_name> &
rule=<rule_name>
```

For example, the following URL allows you to view a rule, `checkCredit` in ruleset `rs2` with a single URL:

```
http://localhost:8888/ruleauthor/ConnectRepos.uix?event=dlConnect&ReposType=File&o
racle.rules.sdk.store.jar.path=C:\jartest\test1.jar&Dict=c233&Version=v1&RuleSet=r
s2&rule=checkCredit
```

Table C–2 URL Parameters for Accessing a Rule Using a URL

URL Component	Description
host_name	Specifies the host where the Rule Author application is running.
port_number	Specifies the port number where the Rule Author application is running.
repos_prop i	Denotes the repository connection property *i* defined by the Oracle Business Rules SDK. Consult the Javadoc for additional details. For file type repository, see oracle.rules.sdk.store.jar.Constants.java. For WebDAV, see oracle.rules.sdk.store.webdav.Constants.java.
repos_value val	Denotes repository connection value *val* for property *i*
dict_version	Specifies the dictionary name
ruleset_name	Optional - specifies the ruleset name. If you do not provide either a <*ruleset_name*> or a <*rule_name*>, the ruleset summary page is shown.
rule_name	Optional - specifies the rule name. If a <*ruleset_name*> is set, but <*rule_name*> is not given, the ruleset page is shown.

C.8 How Do I Use a Property Change Listener in Oracle Business Rules?

The Oracle Rules Engine supports the Java PropertyChangeListener design pattern. This allows an instance of a Java fact that uses the PropertyChangeSupport class to automatically notify the Oracle Rules Engine when property values have changed. Java facts are not required to implement this pattern to be used by Oracle Rules Engine.

Normally, changes made to values of a property of a Java object that has previously been asserted to the Oracle Rules Engine requires that the object be re-asserted in order for rules to be reevaluated with the new property value. For properties that fire PropertyChangeEvents, changing the value of those properties both changes the value and re-asserts the fact to the Oracle Rules Engine.

To implement the PropertyChangeListener design pattern in a class, do the following:

1. Import this package in the class:

```
import java.beans.PropertyChangeSupport;
```

2. Add a private member variable to the class:

```
private PropertyChangeSupport m_pcs = null;
```

3. In the constructor, create a new Property Change Support object:

```
m_pcs = new PropertyChangeSupport(this);
```

4. Then for each setter, add the call to firePropertyChange:

```
public void setName( String name ){
   String oldVal =  m_name;
   m_name = name;
   m_pcs.firePropertyChange( "name", oldVal, m_name );
}
```

5. Implement addPropertyChangeListener method (delegate to m_pcs):

```
public void addPropertyChangeListener(PropertyChangeListener pcl){
     m_pcs.addPropertyChangeListener( pcl );
}
```

6. Implement removePropertyChangeListener method (delegate to m_pcs):

```
public removePropertyChangeListener(PropertyChangeListener pcl){
   m_pcs.removePropertyChangeListener( pcl );
}
```

When deciding whether to design your application to always explicitly re-assert modified objects or implement the PropertyChangeListener design pattern, consider the following items:

- Explicitly re-asserting modified objects allows a user to group several property changes and making them visible to the rules all at once. This is most useful when a concurrent thread is executing rules, and the rules should see only a complete group of property changes.
- Explicit assert reduces the computational cost of rule re-evaluation when multiple properties are changed. If multiple properties are changed at the same time, this will result in multiple re-evaluations of rule conditions that reference the fact type. This occurs because each property change event results in a re-assertion of the

object. Using an explicit assert instead of the PropertyChangeListener pattern will eliminate this extra computational cost.

- Explicit assert is required when a rule modifies a fact that is also tested in its condition, but the automatic reassert triggered by the PropertyChangeListener before a guard condition property is set would cause the rule to refire itself endlessly
- Explicit assert must be used when modifying RL facts and XML facts, since these cannot be defined to support the PropertyChangeListener design pattern.
- PropertyChangeListener-enabled facts allow a Java application to communicate property changes to the rule engine without having to change the application to perform explicit asserts. This also means that code that modifies a property of an object does not need to have a reference to the RuleSession object in scope
- PropertyChangeListener support prevents the common error of neglecting to re-assert a fact after changing its properties.

C.9 How Do I Modify the Single Sign-On Timeout for Oracle Application Server?

If you find that the default Oracle Single Sign-On timeout specified for your applications, including Rule Author is too low for your installation, you can change the value specified for this timeout.

Perform the following steps to change the Single Sign-On timeout:

1. In Application Server Control, go to the Cluster Topology page.
2. In the Administration area, click **Java SSO Configuration**.
3. In the Properties area, change the value in the **Session Timeout (secs)** text box to the desired timeout value.
4. Click **Apply**.

See Also: Section A.2, "Rule Author Session Timeout"

C.10 Does Oracle Business Rules Provide High Availability?

If you need to run a rules application as a highly available application, you should refer to the Oracle Application Server High Availability Guide for information on Oracle Business Rules and high availability.

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:

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 and Checking 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. Likewise, when using XML schema, the generated JAXB classes need to be in the classpath. For more information about setting the OC4J classpath, see Section 3.11, "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.

D.11 Changing Language When Using Rule Author

To change the Rule Author display language, you need to log out, change the language, and then log back in to Rule Author.

If you do not log out, you may see a `java.lang.NullPointerException` error.

The procedure required to change the language is:

1. Log out of Rule Author.
2. Change the language in the browser.
3. Log back in to Rule Author

D.12 Why Do I Get a File Error When Simultaneously Editing and Executing a Ruleset Under Microsoft Windows?

On Microsoft Windows operating systems, a file in use by one application may not be used by another application. This means it is possible for an error to occur when Rule Author attempts to write to a local file repository while an application is attempting to read from the same repository. This should be a rare occurrence because of the small time windows involved.

The signature for this type of failure looks similar to the following:

```
oracle.rules.sdk.store.StoreException: Unable to rename
 '<your-repository-file-name>' so that it can be replaced.
at oracle.rules.sdk.store.jar.JarStore.writeJar(JarStore.java:752)
at oracle.rules.sdk.store.jar.JarStore.flush(JarStore.java:211)
at oracle.rules.sdk.repository.impl.RuleRepositoryImpl._
flushChanges(RuleRepositoryImpl.java:381)
at oracle.rules.sdk.repository.impl.RuleRepositoryImpl._
save(RuleRepositoryImpl.java:367)
at
oracle.rules.sdk.repository.impl.RuleRepositoryImpl.save(RuleRepositoryImpl.java:2
65)
at
oracle.tip.tools.ide.rules.ide.jdeveloper.JDevRulesProject.saveDictionary(JDevRule
sProject.java:83)
```

This can be caused by another rules based application reading the repository at the same time as this attempt to update it. The operation may be retried. If this error persists, then some other application is keeping the repository open.

To workaround this issue, simply retry the operation.

D.13 Why are Ancestor Methods not Visible from Sub-Classes

The properties of a superclass are visible in the appropriate choice lists, but the methods of the ancestor classes are not visible.

There is no workaround for this issue.

D.14 Adding an XML Schema Results in Error RUL-01627

Sometimes when you attempt to add a schema Rule Author reports the following error:

```
Cannot perform operation. 'RUL-01627
The schema X has been imported.
Please delete it if you want to reimport the same schema
```

When you add the schema is not displayed in the schema path box, so it appears that either the list is incorrect or the error is incorrect. The workaround for this problem is to reload the XMLFact page, then add the schema again.

D.15 Choice List with Client and Server Using Different Locale Generates Invalid RL Language

The choice list entry that denotes "nothing selected" is translated. The translated value is controlled by the locale set in the RuleDictionary. There are two usage cases in which this can trigger incorrect behavior in the Rules SDK.

- The `RuleDicitonary` has a `setLocale` method. In a client/server usage if the client and server are not using the same locale and the server locale is not English, then failure to use setLocale to set the locale in the client can result in the generation of invalid RL Language.
- If the server locale is not English and the application never invokes a `setX` (where X is the property name for the list, then the Rules SDK does not correctly set the unselected entry back to the canonical form (which is English). This results in generated that does not compile correctly (usually due to missing values)

To work around these issues:

When an application may be running in an environment where the client and server may be using different locales or the server may not be an English locale, the application should use the `RuleDictionary setLocale` to set the locale to that of the client, if lists are being displayed, and all lists should be set. For example, this problem might occur when a Expression Form property is not set explicitly by the application.

D.16 Invalid RL Language Generated When Inherited Classes are Used

In some cases invalid RL Language is generated for a rule set. The following error is reported when you attempt to execute this code:

```
A syntax error is found.
Error:fact class should be declared earlier at line X column Y in rulesetZ
```

This error occurs when two Java classes that have an inheritance relationship are used in a ruleset and the child class is used before the parent class. This use can be in a rule condition, an `assert`, a `retract`, or an `assertXPath`. If a subclass or interface is referenced in one of these contexts before the super class is, then a reference to the super class in one of these contexts results in a `FactClassException` when the RL Language is interpreted.

Rule Author does not generate the correct RL Language, a `fact class` statement, to resolve this issue when the `supports xpath` attribute for each Java class involved is set to `false` in the datamodel (this is the default).

To workaround this issue, the `supports xpath` attribute for each Java class involved should be set to `true` in the datamodel. In Rule Author, this is accomplished using a checkbox when viewing the Java class. For some classes, this workaround may not generate correct RL Language because the `fact class` statements are not generated in the correct order. In this case, there is no workaround.

Some inheritance hierarchies are not allowed in RL Language, specifically ones that require multiple inheritance. For instance, if `Interface2` extends `Interface1`, and `Class1` directly implements both interfaces, then a single inheritance tree cannot be determined. If `Class1` only implemented `Interface2`, then a single inheritance tree can be determined and the classes used in RL Language.

See Also: "Viewing Java Objects in a Data Model" on page 3-9

Index

S

T

U

V

W

X