Oracle® Business Rules

ユーザーズ・ガイド

10*g*（10.1.3.1.0）

部品番号 : B31866-02

2007 年 2 月

ORACLE®

Oracle Business Rules ユーザーズ・ガイド , 10*g*（10.1.3.1.0）

部品番号 : B31866-02

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

原本部品番号 : B28965-02

原本著者 : Thomas Van Raalte

原本協力者 : Kevin Yu Hwang, Qun Chen, Ching Luan Chung, David Clay, Kathryn Gruenefeldt, Gary Hallmark, Phil Varner, Neal Wyse, Lance Zaklan

目次

はじめに vii

対象読者 viii
ドキュメントのアクセシビリティについて viii
関連ドキュメント viii
表記規則 ix
サポートおよびサービス ix

1 Oracle Business Rules の概要

1.1 Oracle Business Rules の概要 1-2
1.1.1 ビジネス・ルールとは 1-2
1.1.2 データ・モデルとは 1-3
1.1.3 ルールベース・システムとは 1-3
1.2 Oracle Business Rules のコンポーネント 1-4
1.2.1 Oracle Business Rules Rule Author の概要 1-4
1.2.2 Oracle Business Rules Rules SDK の概要 1-5
1.2.3 Oracle Business Rules RL Language の概要 1-5
1.2.4 Oracle Business Rules Rules Engine の概要 1-6
1.3 Oracle Business Rules Rule Author の用語と概念 1-6
1.3.1 ルールの使用 1-6
1.3.2 ルールセットの使用 1-7
1.3.3 リポジトリとディクショナリの使用 1-7
1.3.4 ファクトの使用 1-7
1.3.5 関数の変数と制約の使用 1-8
1.4 Java アプリケーションをルール対応にする手順 1-9
1.4.1 ルール対応の対象となるアプリケーション領域の識別 1-9
1.4.2 データ・モデル用の Rule Author 定義の準備 1-10
1.4.3 データ・モデルに関するビジネス用語の開発 1-10
1.4.4 ルールの記述とカスタマイズ 1-10
1.4.5 Oracle Rules Engine を使用するアプリケーション・ロジックの変更または作成 1-10
1.4.6 ルール対応アプリケーションのテスト 1-10

2 Rule Author の開始

2.1 Rule Author ユーザーの作成 2-2
2.2 Rule Author の起動 2-3
2.3 Rule Author のホーム・ページ 2-4
2.4 レンタカー・サンプルに対するディクショナリの作成および保存 2-5
2.4.1 Rule Author リポジトリへの接続 2-5
2.4.2 Rule Author ディクショナリの作成 2-7
2.4.3 Rule Author ディクショナリのバージョン付き保存 2-8
2.4.4 Rule Author ディクショナリの保存 2-8
2.5 レンタカー・サンプルに対するデータ・モデルの定義 2-9
2.5.1 レンタカー・サンプルでのファクトとしての Java オブジェクトの使用 2-9
2.5.2 Rule Author への Java クラスおよびパッケージの追加 2-9
2.5.3 データ・モデルへの Java クラスのインポート 2-11
2.5.4 現状の定義の保存 2-12
2.6 レンタカー・サンプルに対するビジネス用語の定義 2-13
2.6.1 Java ファクト定義に対するビジネス用語の指定 2-13
2.6.2 関数に対するビジネス用語の指定 2-14
2.6.3 プロパティとメソッドに対する参照可能性の指定 2-14
2.7 レンタカー・サンプルに対するルールの定義 2-15
2.7.1 レンタカー・サンプルに対するルールセットの作成 2-15
2.7.2 レンタカー・サンプルに対するルールの作成 2-16
2.8 レンタカー・サンプルに対するルールのカスタマイズ 2-24
2.9 Oracle Business Rules を使用した Java アプリケーションの作成 2-25
2.9.1 Rules SDK および Rules RL Language クラスのインポート 2-25
2.9.2 Rules SDK でのリポジトリの初期化 2-26
2.9.3 Rules SDK でのディクショナリのロード 2-26
2.9.4 ルールセットの指定と Rules SDK での RL Language の生成 2-27
2.9.5 ルール・セッションの初期化および実行 2-27
2.9.6 ルール・セッション内のビジネス・オブジェクトのアサート 2-28
2.9.7 ルール・セッションでの実行関数の使用 2-28
2.10 テスト・プログラムを使用したレンタカー・サンプルの実行 2-29

3 Rule Author 機能の使用

3.1 変数の使用 3-2
3.2 制約の使用 3-3
3.3 RL ファクトの使用 3-5
3.4 関数の使用 3-7
3.5 ルールの使用 3-9
3.6 データ・モデル内の Java オブジェクトの表示 3-9
3.6.1 Rule Author のリストに関する参照可能性およびオブジェクト・チェーンの指定 3-10
3.7 Oracle Business Rules RL Language テキストの生成 3-11
3.7.1 RL Language テキストの生成、表示およびチェック 3-11
3.8 Rule Author のディクショナリ・プロパティの構成 3-12
3.8.1 「アドバンスト・テスト式」オプションの使用 3-13
3.8.2 「ロギング」オプションの使用 3-13

3.9 Rule Author のディクショナリの削除 ... 3-14
3.10 ディクショナリのインポートとエクスポート ... 3-14
3.11 テスト・ルールセットの使用 ... 3-16
3.12 ルールの起動と Rules Engine 結果の取得 ... 3-19
3.12.1 結果の例の概要 ... 3-19
3.12.2 グローバル変数を使用した結果の取得 ... 3-20
3.12.3 コンテナ・オブジェクトを使用した結果の取得 ... 3-21
3.12.4 判断オブジェクトを使用した結果の取得 ... 3-22

4 Rule Author での XML ファクトの使用

4.1 Rule Author での XML 文書およびスキーマの使用の概要 ... 4-2
4.2 Rule Author ユーザーの作成と Rule Author の起動 ... 4-2
4.3 XML 版レンタカー・サンプルに対するディクショナリの作成および保存 ... 4-3
4.3.1 Rule Author リポジトリへの接続 ... 4-3
4.3.2 Rule Author ディクショナリの作成 ... 4-5
4.3.3 Rule Author ディクショナリのバージョン付き保存 ... 4-6
4.3.4 Rule Author ディクショナリの保存 ... 4-6
4.4 XML 版レンタカー・サンプルに対するデータ・モデルの定義 ... 4-7
4.4.1 XML 版レンタカー・サンプルでのファクトとしての XML スキーマの使用 ... 4-7
4.4.2 レンタカー・サンプルに対する XML ファクトの追加（XML スキーマ処理） ... 4-7
4.4.3 データ・モデルへの XML スキーマ要素のインポート ... 4-10
4.4.4 データ・モデルでの XML ファクトの表示 ... 4-11
4.4.5 現状の XML ファクト定義の保存 ... 4-12
4.5 XML 版レンタカー・サンプルに対するビジネス用語の定義 ... 4-12
4.5.1 XML ファクト定義に対するビジネス用語の指定 ... 4-12
4.5.2 関数に対するビジネス用語の指定 ... 4-13
4.5.3 XML ファクトのプロパティとメソッドに対する参照可能性の指定 ... 4-14
4.6 XML 版レンタカー・サンプルに対するルールの定義 ... 4-14
4.6.1 XML 版レンタカー・サンプルに対するルールセットの作成 ... 4-14
4.6.2 XML 版レンタカー・サンプルに対するルールの作成 ... 4-15
4.7 XML 版レンタカー・サンプルに対するルールのカスタマイズ ... 4-23
4.8 XML ファクトを使用したルール・セッションがある Java アプリケーションの作成 ... 4-24
4.8.1 Rules SDK および Rules RL クラスのインポート ... 4-25
4.8.2 JAXB コンテキストの作成と XML 文書のアンマーシャリング ... 4-25
4.8.3 Rules SDK でのリポジトリの初期化 ... 4-25
4.8.4 Rules SDK でのディクショナリのロード ... 4-26
4.8.5 ルールセットのロードおよびデータ・モデルとルールセットに対する RL Language の生成 ... 4-26
4.8.6 ルール・セッションの初期化および実行 ... 4-27
4.8.7 ルール・セッション内からの XML データのアサート ... 4-27
4.8.8 ルール・セッションでの実行関数の使用 ... 4-28
4.9 テスト・プログラムを使用した XML 版レンタカー・サンプルの実行 ... 4-28

5 JSR-94 の使用

5.1 Oracle Business Rules での JSR-94 ルール実行セットの使用 5-2
5.1.1 ファイル・リポジトリのルールセットからの JSR-94 ルール実行セットの作成 5-2
5.1.2 WebDAV リポジトリからの JSR-94 ルール実行セットの作成 5-3
5.1.3 Oracle Business Rules RL Language テキストからのルール実行セットの作成 5-4
5.1.4 URL で指定した RL Language テキストからのルール実行セットの作成 5-6
5.1.5 複数のソースからのルールセットを持つルール実行セットの作成 5-7
5.2 JSR-94 インタフェースを Oracle Business Rules とともに使用 5-7
5.2.1 CreateRuleExecutionSet を使用したルール実行セットの作成 5-7
5.2.2 createRuleSession を使用したルール・セッションの作成 5-8
5.2.3 JSR-94 メタデータの使用 5-8
5.2.4 Oracle Business Rules の JSR-94 拡張機能の使用 5-8

6 Oracle Business Rules SDK の使用

6.1 Rules SDK の構築ブロック 6-2
6.2 リポジトリとディクショナリの使用 6-2
6.2.1 WebDAV リポジトリへの接続の確立 6-3
6.2.2 ファイル・リポジトリへの接続の確立 6-3
6.2.3 ディクショナリのロード 6-3
6.3 データ・モデルの使用 6-4
6.3.1 データ・モデルの作成 6-5
6.3.2 データ・モデル・コンポーネントの作成 6-5
6.3.3 関数の引数リストの作成 6-5
6.3.4 初期化式の作成 6-6
6.3.5 RL 関数本体の作成 6-6
6.4 ルールセットの使用およびルールの作成と変更 6-7
6.4.1 ルールセットの作成 6-8
6.4.2 ルールセットへのルールの追加 6-8
6.4.3 ルールへのパターンの追加 6-9
6.4.4 パターンへのテストの追加 6-9
6.4.5 ルールへのアクションの追加 6-10
6.4.6 ルールセットおよびルールの追加に関する注意事項 6-11

A Oracle Business Rules のファイルと制限事項

A.1 Rule Author のネーミング規則 A-2
A.1.1 ルールセットのネーミング A-2
A.1.2 ディクショナリのネーミング A-2
A.1.3 バージョンのネーミング A-2
A.1.4 別名のネーミング A-2
A.1.5 XML スキーマのターゲット・パッケージのネーミング A-2
A.2 Rule Author のセッション・タイムアウト A-3
A.3 Rules SDK と Rule Author のテンポラリ・ファイル A-3

B リポジトリを利用した Rule Author と Rules SDK の使用

B.1 WebDAV リポジトリの使用 B-2
B.1.1 WebDAV リポジトリの設定 B-2
B.1.2 WebDAV リポジトリへの接続 B-3
B.1.3 プロキシを使用した WebDAV リポジトリへの接続 B-3
B.2 WebDAV リポジトリのセキュリティ B-4
B.2.1 Rule Author の SSL を経由した WebDAV リポジトリとの通信 B-4
B.2.2 Oracle ウォレットの場所の設定 B-4
B.2.3 WebDAV リポジトリ認証に対する Rule Author の構成 B-5
B.2.4 Oracle ウォレットへの WebDAV リポジトリ認証に関するデータの格納 B-5
B.3 ファイル・リポジトリの使用 B-6
B.3.1 ファイル・リポジトリの設定 B-6
B.3.2 ファイル・リポジトリの更新とテンポラリ・ファイル B-7
B.4 リポジトリの高可用性 B-7

C Oracle Business Rules に関するよくある質問

C.1 ルール運用に関する質問 C-2
C.1.1 ルール・アクションのファクトの状態が、ルール条件と矛盾しているのはなぜでしょうか。... C-2
C.1.2 変更した Java オブジェクトをファクトとしてアサートしましたが、ルールが起動しません。なぜでしょうか。..... C-2
C.1.3 Oracle Business Rules RL Language と Java の相違は何ですか。..... C-2
C.1.4 Rules SDK を使用して式に NULL を含めるにはどうすればよいですか。..... C-3
C.2 Oracle Business Rules を使用するには、どのような JAR ファイルが必要ですか。..... C-3
C.3 Rule Author を Oracle 以外のコンテナにデプロイするにはどうすればよいですか。..... C-4
C.3.1 WebSphere V6.1（WAS V6.1）への Rule Author のデプロイ C-4
C.3.2 WebLogic Server への Rule Author のデプロイ C-7
C.3.3 JBoss 4.0 への Rule Author のデプロイ C-8
C.4 RuleSession では並行性と同期化はどのように処理されますか。..... C-10
C.5 Oracle Business Rules のランタイム・パフォーマンスを向上させるにはどうすればよいですか。... C-12
C.6 RL Language のクロス積の正しい使用方法を教えてください。..... C-13
C.7 URL を使用してディクショナリのルールにアクセスするにはどうすればよいですか。..... C-14
C.8 Oracle Business Rules ではプロパティ変更リスナーをどのように使用しますか。..... C-15
C.9 Oracle Application Server での Single Sign-On タイムアウトを変更するにはどうすればよいですか。..... C-16
C.10 Oracle Business Rules に高可用性は備わっていますか。..... C-17

D Oracle Business Rules のトラブルシューティング

D.1 Rule Author でアクセスできないパブリック・ファクト変数 D-2
D.2 RL 関数で使用できないグローバル変数 D-2
D.3 JDK 1.4.2 クラスのインポート D-2
D.4 Firefox 上のポップアップ・ウィンドウの管理 D-3
D.5 メソッドでの文字列データ型の使用 D-3
D.6 データ・モデル内のクラス順序と階層の保持 D-3
D.7 Rule Author から生成された RL の検証およびチェック D-4
D.8 Java パッケージ名の一部としての RL 予約語の使用 D-4
D.9 非表示の getter メソッドと setter メソッド D-4
D.10 実行時にアサートされない XML ファクト D-4
D.11 Rule Author の使用時の言語の変更 D-4

D.12 Microsoft Windows でルールセットの編集と実行を同時に行う場合にファイル・エラーが発生する理由 D-5
D.13 上位メソッドをサブクラスから表示できない理由 D-5
D.14 XML スキーマを追加すると、エラー RUL-01627 が発生します。..... D-5
D.15 クライアントとサーバーが異なるロケールを使用している選択リストによる無効な RL Language の生成 D-6
D.16 継承されたクラスの使用時に生成される無効な RL Language D-6

索引

はじめに

ここでは、次の項目について説明します。

- 対象読者
- ドキュメントのアクセシビリティについて
- 関連ドキュメント
- 表記規則
- サポートおよびサービス

対象読者

このマニュアルは、次の作業を行うアプリケーション・プログラマ、システム管理者およびその他のユーザーを対象としています。

- Oracle Business Rules プログラムの作成
- 既存の Oracle Business Rules プログラムの変更およびカスタマイズ
- ルール・プログラムを使用した Java アプリケーションの作成
- 既存の Java アプリケーションへのルール・プログラムの追加

このマニュアルを使用するには、Java プログラミング言語の基本的な作業知識が必要です。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし一部のスクリーン・リーダーは括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。

関連ドキュメント

リリース・ノート、インストール関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J（Oracle Technology Network Japan）から、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。登録は、次の Web サイトから無償で行えます。

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

すでに OTN-J のユーザー名およびパスワードを取得している場合は、次の URL で OTN-J Web サイトのドキュメントのセクションに直接接続できます。

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

表記規則

このマニュアルの本文では、次の表記規則を使用します。

規則	意味
太字	太字は、処理に関連付けられているグラフィカル・ユーザー・インタフェース（GUI）要素、または本文中や用語集で定義されている用語を示します。
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル・コード、画面に表示されるテキストまたは入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

```
http://otn.oracle.co.jp/document/
```

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

```
http://www.oracle.co.jp
http://otn.oracle.co.jp
```

> **注意：** ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

Oracle Business Rules の概要

このマニュアルには、Oracle Business Rules の使用方法に関する情報が記載されています。Oracle Business Rules は Oracle Application Server の 1 つのコンポーネントで、アプリケーションを規制や競争上の要求に対し迅速に対応できるようにします。このような適応性が増すのは、Oracle Business Rules を使用することで、ビジネス・アナリストがビジネス・ルールをアプリケーション・コードから切り離して作成および変更できるためです。Oracle Business Rules を使用すると、ビジネス・アナリストは、ビジネス・プロセスを停止することなくビジネス・ルールを変更できます。また、ビジネス・ルールを外部的なものにすることによって、ビジネス・アナリストはプログラマの介入なしにビジネス・ルールを直接管理できます。

このマニュアルでは、Oracle Business Rules Rule Author（Rule Author）の使用方法、Oracle Business Rules SDK（Rules SDK）の説明、およびルール対応 Java プログラムの作成方法を説明します。

この章の内容は次のとおりです。

- Oracle Business Rules の概要
- Oracle Business Rules のコンポーネント
- Oracle Business Rules Rule Author の用語と概念
- Java アプリケーションをルール対応にする手順

1.1 Oracle Business Rules の概要

この項では、ビジネス・ルールの概念について説明します。内容は次のとおりです。

- ビジネス・ルールとは
- データ・モデルとは
- ルールベース・システムとは

1.1.1 ビジネス・ルールとは

ビジネス・ルールとは、ビジネス・ポリシーを記述した文のことです。たとえば、レンタカー会社であれば、次のようなビジネス・ルールを使用します。

If the age of a driver is younger than 21, then decline to rent.
（運転者の年齢が 21 歳未満の場合、貸し出しを拒否する。）

航空会社であれば、次のようなビジネス・ルールを使用します。

If a frequent flyer account has total miles for the year that are greater than 100,000, then status is Gold.
（頻度の高い飛行機利用者の年間飛行距離合計が 100,000 マイルを超える場合、ゴールド・ステータスとする。）

金融機関であれば、次のようなビジネス・ルールを使用します。

If annual income is less than $10,000, then deny loan.
（年間収入が 10,000 ドル未満の場合、貸付を拒否する。）

これらの例は、それぞれ個別のビジネス・ルールを示しています。実際には、Oracle Business Rules を使用して、多数のビジネス・ルールを組み合せることができます。

たとえば、レンタカー会社の例では、運転者の年齢に関するルールに「UnderAge ルール」という名前を指定できます。UnderAge ルールなどのルールは、従来、アプリケーション・コードに組み込まれ、Java アプリケーションでは、次のように表示されていました。

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

技術的な知識のないユーザーが、このようなコードを読んで理解し、変更することは困難です。たとえば、レンタカー会社がポリシーを「18 歳未満」に変更して、18 歳未満のすべての運転者を UnderAge ルールの対象にするとします。多くの本番環境では、開発者がアプリケーションを変更し、変更したアプリケーションを再コンパイルして再デプロイする必要があります。Oracle Business Rules を使用して作成したビジネス・ルール・アプリケーションでは、ビジネス・ルールが容易に変更できるようにサポートされるため、このプロセスが簡素化されます。

Oracle Business Rules を使用すると、ビジネス・アナリストは、ほとんどプログラマの援助なしに、ビジネス・ルールで表現されたビジネス・ポリシーを変更できます。Oracle Business Rules を使用する**ルール対応アプリケーション**と呼ばれるアプリケーションでは、新しい政府規制や、社内プロセスの改善、顧客とサプライヤの関係の変化にアプリケーションが対応できるよう、継続的な変更がサポートされます。

関連項目： 1-9 ページ「Java アプリケーションをルール対応にする手順」

1.1.2 データ・モデルとは

Oracle Business Rules では、**ファクト**とは、Rules Engine 内でアサートされるデータ・オブジェクトです。UnderAge ルールなどのルールは、ファクトを制約し、サポートします。Oracle Business Rules では、**データ・モデル**とは、ビジネス・ルールの作成に使用できるファクトまたはビジネス・オブジェクトのタイプを指定するものです。たとえば、レンタカー会社で運転者の年齢と照合させるルールを作成する場合は、運転者に関する情報が、ルールに使用されるファクトとなります。Rule Author を使用すると、データ・モデルを定義し、そのデータ・モデル内でオブジェクトを使用してルールを作成できます。

1.1.3 ルールベース・システムとは

この項の内容は次のとおりです。

- Rete アルゴリズムを使用したルールベース・システム
- Oracle Business Rules のルールベース・システム

1.1.3.1 Rete アルゴリズムを使用したルールベース・システム

Rete アルゴリズムは、人工知能の研究者によって 1970 年代後期に最初に開発され、現在では、様々なベンダーが提供する Rules Engine の核となっています。Oracle Business Rules では、ルールとファクトのパターン一致プロセスを最適化するために、この Rete アルゴリズムを使用しています。Rete アルゴリズムは、部分一致の結果を、現行の作業メモリーにあるノードの単一ネットワークに格納します。

Rete アルゴリズムを使用することで、Rules Engine では、ファクトの削除、追加または変更時に不要な再確認を回避できます。ファクトとルールを処理するために、Rete アルゴリズムは、各ファクト定義に入力ノードを、各ルールに出力ノードを作成し、使用します。ファクト参照は、入力ノードから出力ノードまでのフローです[1]。

Rete アルゴリズムの利点は、次のとおりです。

- ルール順序からの独立 : 他のルールに影響を与えずに、ルールを追加および削除できます。
- 複数ルール間での最適化 : 共通の条件を使用する各ルールは、Rete ネットワーク内でノードを共有します。
- 高いパフォーマンスの推論サイクル : 通常、各ルールの起動によって変更されるのは少数のファクトのみで、Rete ネットワークの更新コストは、ファクトまたはルールの総数ではなく、変更されたファクトの数に比例します。

1.1.3.2 Oracle Business Rules のルールベース・システム

Rete アルゴリズムを使用したルールベース・システムは、Oracle Business Rules の基盤です。ルールベース・システムは、次の要素で構成されています。

- ルールベース : 適切なビジネス・ポリシー、およびその他の If-Then ルールにエンコードされたナレッジ。
- 作業メモリー : システムに追加された情報。Oracle Business Rules では、アサート・コールを使用してシステムに一連のファクトを追加します。
- 推論エンジン : ルールを処理する Rules Engine。一連のファクトを対象とした特定の実行において、ファクトに適合するルールを、パターン一致を実行して判断します。

Oracle Business Rules では、ルールベース・システムはデータドリブンの**順連鎖化システム**です。起動するルールはファクトによって決定されます。一連のファクトに一致するルールが起動すると、そのルールによって新しいファクトが追加される場合があります。これらの新しい

[1] ファクト参照は、入力ノードから出力ノードまでのフローです。入力ノードと出力ノード間にはテスト・ノードと結合ノードがあります。テストは、ルール条件にブール式がある場合に発生します。結合は、ルール条件 AND によって 2 つのファクトが結合される場合に発生します。ルールがアクティブ化されるのは、その出力ノードにファクト参照が含まれている場合です。ファクト参照は、アクティブ化されたルールの再計算を速めるために、ネットワーク全体にキャッシュされます。ファクトが追加、削除または変更されると、Rete ネットワークは、キャッシュとルールのアクティブ化を更新します。これに必要なのは、作業の増分のみです。

ファクトは再びルールに対して実行されます。このプロセスは、結論に達するか、このサイクルが停止またはリセットされるまで繰り返し実行されます。このように、順連鎖化のルールベース・システムでは、ファクトによってルールが起動し、ルールの起動によって新たなファクトが作成され、そのファクトによってさらに別のルールが起動します。このようなプロセスを、**推論サイクル**と呼びます。

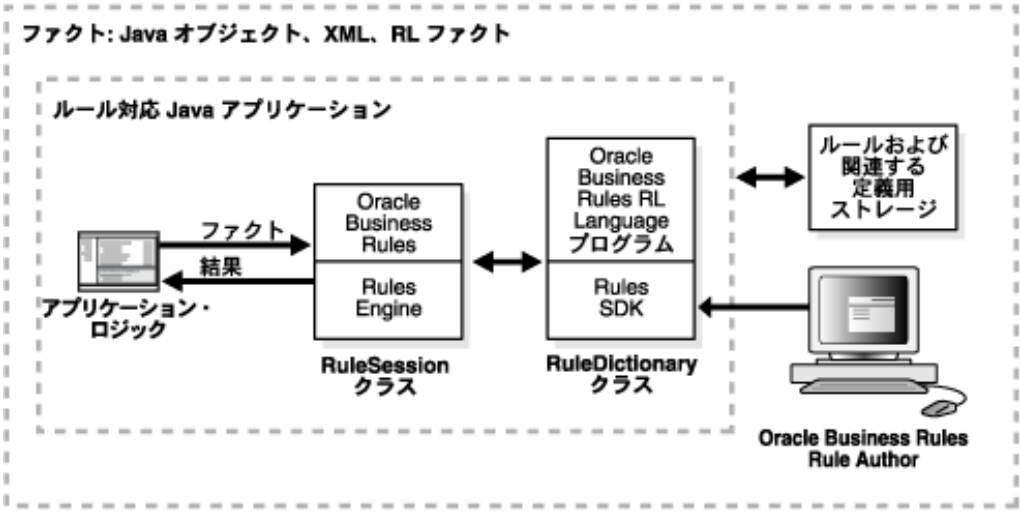
1.2 Oracle Business Rules のコンポーネント

図 1-1 に、Oracle Business Rules の各コンポーネントを示します。

内容は次のとおりです。

- Oracle Business Rules Rule Author の概要
- Oracle Business Rules Rules SDK の概要
- Oracle Business Rules RL Language の概要
- Oracle Business Rules Rules Engine の概要

図 1-1 Oracle Business Rules のアーキテクチャ



1.2.1 Oracle Business Rules Rule Author の概要

Oracle Business Rules Rule Author（Rule Author）を使用すると、Web ブラウザを使用してどこからでもルールを操作できます。これは、新規ルールの作成と既存ルールの編集用の point-and-click インタフェースを提供します。Rule Author を使用すると、ビジネス・ルールとデータ・モデルを直接操作できます。Rule Author の操作に、Oracle Business Rules RL Language（RL Language）の理解は不要です。Rule Author には、ビジネス・ルールを簡単に作成、表示および変更するための方法が用意されています。

Rule Author は、アプリケーション開発者やビジネス・アナリストなど、様々なタイプのユーザーをサポートしています。アプリケーション開発者は、Rule Author を使用して、データ・モデルとルールの初期セットを定義します。ビジネス・アナリストは、Rule Author を使用して、ルールの初期セットを操作したり、ビジネス・ニーズに従ってルールの初期セットを変更およびカスタマイズします。Rule Author を使用すると、ビジネス・アナリストは、ほとんどプログラマの援助なしに、ルールを作成およびカスタマイズできます。

Rule Author は、ルール・プログラムをディクショナリに格納します。このディクショナリは、Rule Author のディクショナリ・ストレージ・プラグインを使用してリポジトリに保存されます。ディクショナリは必要な数だけ作成でき、各ディクショナリには複数のバージョンを指定できます。ルール対応プログラムは、Oracle Business Rules SDK を使用してディクショナリにアクセスします。

出荷時の Rule Author では、WebDAV（Web Distributed Authoring and Versioning）リポジトリとファイル・リポジトリがサポートされています。

注意 1: 複数のユーザーが同じディクショナリを編集することは安全ではありません。

注意 2: ファイル・リポジトリの場合、所定の時間にリポジトリを編集できるのは、そのリポジトリに格納されているディクショナリ数に関係なく常に 1 人のユーザーのみです。WebDAV リポジトリの場合は、1 人のユーザーが複数のディクショナリを同時に編集できます。

1.2.2 Oracle Business Rules Rules SDK の概要

Oracle Business Rules SDK（Rules SDK）は、ビジネス・ルールの管理機能を提供する Java ライブラリで、開発者は、カスタマイズしたルール・プログラムを記述するために使用できます。Rule Author では、この Rules SDK を使用し、適切に定義されたインタフェースを介して、ルールとデータ・モデルを作成および変更し、ルールとデータ・モデルにアクセスします。カスタム・アプリケーションでは、Rules SDK を使用して、ルールとデータ・モデルのコレクションを表示、作成および変更できます。

Rules SDK API をルール対応アプリケーションで使用して、ルールへのアクセスや、ルールの作成と変更ができます。ルールおよび関連するデータ・モデルは最初、カスタム・アプリケーションで作成されるか、または Rule Author を使用して作成されます（Rule Author は、Rules SDK ディクショナリ・ストレージ・プラグインを使用してルールを格納します）。

Rules SDK および Rules SDK のディクショナリ・ストレージ・プラグイン部分を使用して、カスタム・リポジトリをサポートすることもできます。

1.2.3 Oracle Business Rules RL Language の概要

Oracle Business Rules は、Oracle Business Rules RL Language（RL Language）と呼ばれる Java に類似した高レベルな言語をサポートしています。RL Language は、Oracle Business Rules プログラムに有効な構文を定義します。RL Language には、Java セマンティクスの能力をサポートするルールを定義するための、Java に類似した直観的な構文が含まれており、使いやすい構文をアプリケーション開発者に提供します。RL Language は、動的に生成したり、ファイルに格納できるテキスト文のコレクションで構成されています。

RL Language を使用すると、アプリケーション・プログラムでは Java オブジェクトをファクトとしてアサートでき、ルールではオブジェクト・プロパティを参照してメソッドを起動できます。同様に、アプリケーション・プログラムでは XML 文書または XML 文書の一部をファクトとして使用できます。

プログラマは、すべての機能を備えたルール・プログラミング言語として RL Language を使用できます。ビジネス・アナリストは、Rule Author を使用してルールを操作できます。この場合、ビジネス・アナリストは、RL Language プログラムを直接表示または記述する必要はありません。

関連項目： RL Language の詳細は、『Oracle Business Rules ランゲージ・リファレンス』を参照してください。

1.2.4 Oracle Business Rules Rules Engine の概要

Oracle Business Rules Rules Engine（Rules Engine）は、ルールをファクトに効率的に適用し、ルールを定義および処理する Java ライブラリです。Rules Engine は、宣言ルール言語を定義し、言語処理エンジン（推論エンジン）を提供し、デバッグのサポート・ツールを提供します。

Rules Engine には、次の機能があります。

- 高いパフォーマンス : Rules Engine は、システムに定義された、ファクトに対して特化された一致アルゴリズムを実装しています。
- パラレル処理アーキテクチャに適したスレッドセーフ実行 : Rules Engine では、あるスレッドがネットワークを評価している間も、別のスレッドでファクトをアサートできます。
- 迅速な対応 : Rules Engine を使用すると、ビジネス・プロセスを停止せずにルールを変更できます。

ルール対応 Java アプリケーションでは、ルール・プログラムをロードして実行できます。ルール対応アプリケーションは、ファクト（Java オブジェクトまたは XML 文書の形式でアサートされたもの）とルールを Rules Engine に渡します。ルール対応 Java アプリケーションで実行される Rules Engine は、Rete アルゴリズムを使用して、ファクトに適したルールを効率的に起動します。

Rules Engine では、RL Language プログラムの開発、テストおよびデバッグに必要な対話型コマンドライン・インタフェースがサポートされています。

1.3 Oracle Business Rules Rule Author の用語と概念

この項では、Rule Author の用語と概念について説明します。内容は次のとおりです。

- ルールの使用
- ルールセットの使用
- リポジトリとディクショナリの使用
- ファクトの使用
- 関数の変数と制約の使用

1.3.1 ルールの使用

ルールは、**If**（条件）と **Then**（アクションのリスト）で構成されています。ルールは簡単な if-then 構造に従っています。この項では、ルールの各構成要素を説明します。

1.3.1.1 ルール条件

ルールの **If** 部分は、ファクトを参照する条件式とルール条件で構成されています。次に例を示します。

If the age of a driver is younger than 21.

条件式はファクト（driver）を参照し、そのファクトのデータ・メンバー（age）が 21 未満であることをテストします。

ルール条件は、ファクトの組合せで条件式が true になると、該当するルールをアクティブ化します。いくつかの点で、ルール条件は、Rules Engine 内の使用可能なファクトへの問合せに類似しています。問合せで戻された各行に対して、ルールがアクティブ化されます。

1.3.1.2 ルール・アクション

ルールの **Then** 部分には、すべてのルール条件が満たされた場合に実行するアクションが記述されています。アクションが実行または起動されるのは、**If** 部分のすべての条件が満たされた場合です。ルールは、いくつかの種類のアクションを実行します。あるアクションでは新しいファクトを追加したり、ファクトを削除できます。また、あるアクションでは Java メソッドを実行したり、RL Language 関数を実行して、ファクトのステータスを変更したり、新しいファクトを作成できます。

ルールは、同時ではなく順番に起動します。ルール・アクションによってルールのアクティブ化設定が変更され、次に起動するルールが変更される場合があることに注意してください。

関連項目： 1-7 ページ「ルールセットの使用」

1.3.2 ルールセットの使用

ルールセットは一連のルールをグループ化します。**ルールセット**は、すべてが一括して評価されるルールのコレクションです。

関連項目： 1-6 ページ「ルールの使用」

1.3.3 リポジトリとディクショナリの使用

Oracle Business Rules で使用するリポジトリには、ディクショナリが格納されます。通常、ディクショナリはルール・アプリケーションに対応し、ルール対応アプリケーション用のルールと定義が格納されます。ディクショナリは、ルールとデータ・モデルが格納されている一連の XML ファイルです。ディクショナリをリポジトリに格納するには、提供されているディクショナリ・ストレージ・プラグインまたはカスタムのディクショナリ・ストレージ・プラグインを使用します。ディクショナリ・ストレージ・プラグイン API は、Rules SDK の一部です。ディクショナリはバージョン化できます。ディクショナリとそのバージョンは、リポジトリに対して、作成、削除、エクスポートおよびインポートができます。

出荷時の Rule Author では、WebDAV（Web Distributed Authoring and Versioning）リポジトリとファイル・リポジトリがサポートされています。

1.3.4 ファクトの使用

Oracle Business Rules では、ルールに対して実行できるファクトは、アサートされたデータ・オブジェクトです。各オブジェクト・インスタンスは単一のファクトに対応しています。オブジェクトが（変更済かどうかに関係なく）再アサートされると、Rules Engine が更新され、そのオブジェクトの新しい状態が反映されます。オブジェクトの再アサートによる新しいファクトの作成はありません。特定のファクト・タイプに複数のファクトを指定するには、オブジェクト・インスタンスを個別にアサートする必要があります。

Rule Author では、データ・モデルの一部であるファクト定義を使用して Oracle Business Rules に、ビジネス・オブジェクトとそのメソッドを認識させます。

この項では、次の 3 種類の Oracle Business Rules ファクト定義を説明します。

- Java ファクト・タイプ定義
- XML ファクト・タイプ定義
- Oracle Business Rules RL Language ファクト・タイプ定義

ルール対応アプリケーションのビジネス・オブジェクトを調査するルールを作成したり、結果をアプリケーションに返す場合は、通常、Java ファクト・タイプと XML ファクト・タイプを使用します。RL Language ファクト・タイプ定義を使用して Rules Engine で他のルールを起動できる中間ファクトを作成します。

1.3.4.1 Java ファクト・タイプ定義

Java ファクト・タイプを使用すると、Java クラスの選択したプロパティとメソッドを Rules Engine に対して宣言できるため、ルールではその Java クラスのインスタンスへのアクセス、作成、変更および削除が可能となります。Java ファクト・タイプを宣言すると、Rules Engine は Java クラスに定義されている public 属性、public メソッドおよび Bean プロパティをアクセスして使用できます（一部のアプリケーションでは Bean プロパティが適切です。これは、Java オブジェクトが `PropertyChangeListener` をサポートしていることが Rules Engine で検出される場合があるためです。この場合は、オブジェクトの変更時に通知するメカニズムが利用されます。）

1.3.4.2 XML ファクト・タイプ定義

XML ファクト・タイプを使用すると、XML 要素の選択した属性とサブ要素または `complexType` を Rules Engine に対して宣言できるため、ルールではそのインスタンスへのアクセス、作成、変更および削除が可能となります。

> **関連項目：** 4-2 ページ「Rule Author での XML 文書およびスキーマの使用の概要」

1.3.4.3 Oracle Business Rules RL Language ファクト・タイプ定義

RL Language ファクト・タイプは、リレーショナル・データベースの行またはメソッドがない JavaBean と同じです。RL Language ファクト・タイプには、RL Language ファクト・タイプ、Java ファクト・タイプ、プリミティブ・タイプのいずれかのメンバーのリストが含まれています。RL Language ファクト・タイプを使用すると、動的な仮想型を提供することで Java アプリケーションのオブジェクト・モデルを拡張できます。

次に例を示します。

```
If customer spent $500 within past 3 months
                then customer is a Gold Customer
```

このルールでは、顧客データを指定する Java ファクト・タイプを使用し、RL Language ファクト・タイプ（Gold Customer）を作成するアクションも使用できます。ルールは、次のように、Gold Customer ファクトを使用するように定義できます。

```
If customer is a Gold customer
                then offer 10% discount
```

このルールでは、Gold Customer という名前の RL Language ファクト・タイプを使用しています。このルールは、Gold Customer ファクトを使用して、過去 3 か月間に 500 ドルの支払があった顧客には 10% 割引の資格があることを示しています。ルールには、顧客が Gold Customer になる他の方法も指定できます。

1.3.5 関数の変数と制約の使用

この項では、次の定義を説明します。

- 関数の定義
- 変数の定義
- 制約の定義

1.3.5.1 関数の定義

Oracle Business Rules では、Java メソッドに類似の方法で関数を定義しますが、RL 関数はクラスに属しません。Java アプリケーション・オブジェクト・モデルは RL 関数を使用して拡張できるため、ユーザーは、元の Java アプリケーション・コードを変更せずに、複数の操作をルールで実行できます。

また、RL 関数定義を使用して、同じまたは類似した式を複数のルール間で共有し、結果をアプリケーションに戻すこともできます。

1.3.5.2 変数の定義

変数の定義を使用すると、複数のルールと関数で情報を共有できます。たとえば、複数のルールで 10% 割引を使用する場合は、変数 Gold Discount を作成し、この変数を使用してすべてのルールに適切な割引を適用できます。

変数の定義を使用すると、プログラムをモジュール化して保守を容易にできます。

1.3.5.3 制約の定義

制約の定義によって、ルールのカスタマイズ可能な部分にマークを付けることができます。たとえば、Gold Customer に提供する割引を、5 から 25 パーセントなどの指定範囲内に制限できます。制約を定義して Rule Author を使用すると、ルール全体の変更を禁止する特別なインタフェースを使用して、指定の範囲内から値を選択できます。

注意： 制約の使用は、ルールのカスタマイズをサポートする Rule Author 機能です（Rule Author の「ルールのカスタマイズ」タブを使用）。

1.4 Java アプリケーションをルール対応にする手順

プログラマとビジネス・アナリストは、協力して Java アプリケーションをルール対応にします。多くの場合、アプリケーションがルール対応になると、プログラマの作業が軽減され、継続中のルール保守はビジネス・アナリストに任されます。

Java アプリケーションのルール対応に必要なタスクは、次のとおりです。

- ルール対応の対象となるアプリケーション領域の識別
- データ・モデル用の Rule Author 定義の準備
- データ・モデルに関するビジネス用語の開発
- ルールの記述とカスタマイズ
- Oracle Rules Engine を使用するアプリケーション・ロジックの変更または作成
- ルール対応アプリケーションのテスト

これらのタスクには、プログラマとビジネス・アナリスト間の協力が必要です。プログラマには、アプリケーション・コードの理解および Java 開発、Web サービス、XML（ビジネス・オブジェクトが XML で表現される場合）の知識が必要です。ビジネス・アナリストには、ビジネス・オブジェクトの高レベルな知識が必要であり、ビジネス・オブジェクトに関する if-then 文としてルールを理解する知識が必要です。ビジネス・アナリストは、頻繁に変更が必要なルールの各部分を確認する必要もあります。

1.4.1 ルール対応の対象となるアプリケーション領域の識別

ビジネス・アナリストとプログラマは協力して、ビジネス・オブジェクトを、ビジネス・ルールでの使用に適したファクトとして公開します。ビジネス・ルールで使用する必要のあるビジネス・ファクトは、ビジネス・アナリストとプログラマで決定します。これらのビジネス・オブジェクトには、頻繁な変更を必要とするポリシーや、変化の早いビジネス・プロセスに対応するために変化するその他のポリシーがあります。

ビジネス・アナリストは、どの機能がルールドリブンとなるかを判別します。たとえば、オンライン・ショッピング・アプリケーションでは、税金と販促に関する機能はルールドリブンとなりますが、ショッピング・カートや製品カタログは対象となりません。

1.4.2 データ・モデル用の Rule Author 定義の準備

プログラマは、Rule Author の定義を使用してデータ・モデルを指定します。プログラマは、ビジネス・アナリストと協力して、有用な関数、中間ファクト、変数および制約も定義する必要があります。

1.4.3 データ・モデルに関するビジネス用語の開発

プログラマとビジネス・アナリストは、Rule Author などのツールを使用して、Rule Author 定義に関するわかりやすいビジネス用語を開発します。これによって、ルールがさらに理解しやすくなります。ビジネス・アナリストは、取得の対象となるビジネス・ファクト、関数および他の定義を確認しながら、ビジネス用語を開発します。

1.4.4 ルールの記述とカスタマイズ

この段階で、ビジネス・アナリストは、定義したビジネス用語を使用し、Rule Author でルールを記述およびカスタマイズできます。また、プログラマは Rules SDK を使用してルールを作成または変更したり、ルール対応アプリケーションの管理部分内のデータ・モデルを作成または変更できます。

1.4.5 Oracle Rules Engine を使用するアプリケーション・ロジックの変更または作成

プログラマは、プロシージャ機能を新規のルールドリブン機能で置換する方法を決定します。アプリケーションが Java で記述されている場合、そのアプリケーション・コードは Rules Engine を直接起動できます。そうでない場合、プログラマは、Web サービスまたは他のリモート API を使用して、Rules Engine を起動する必要があります。プログラマは、新しいアプリケーションを作成するか、既存のアプリケーションを変更して、Rules Engine と対話する必要があります。

注意： ルールに対応しているプロシージャ・コードは、既存のルールをコードから抽出するために利用される場合があります。

関連項目： Rule Author の使用に関する詳細と Java アプリケーションをルール対応にするためにプログラマが実行する手順については、第 2 章「Rule Author の開始」を参照してください。

1.4.6 ルール対応アプリケーションのテスト

プログラマとビジネス・アナリストは、アプリケーションをテストします。プログラマは一連のテストを提供し、また、複雑なルールセットのデバッグ作業を手助けする必要もあります。プログラマは Rules Engine のトレース機能を有効にして、ファクト、ルールのアクティブ化、およびルールの起動に関する情報を提供できます。プログラマは、一連のビジネス・アナリスト・ルールを検証するテスト・ファクトをロードするための自動メカニズムを開発する必要があります。

2

Rule Author の開始

この章では、Oracle Business Rules Rule Author（Rule Author）をチュートリアル形式で紹介します。Rule Author の起動方法、データ・モデルの作成方法およびルールの作成および保存方法について説明します。また、Rules Engine とともに動作する Java サンプル・アプリケーションの作成方法も示します。

このマニュアルでは、レンタカー・サンプルを使用して、Rule Author での作業方法を説明します。レンタカー・サンプルでは、運転者データを使用して運転者情報を指定し、ビジネス・ルールを使用してレンタカー会社のサービス担当が運転者の年齢制限により車両の貸出しを断る必要があるかどうかを判断します。この例を使用して、UnderAge ルールという 1 つのルールを作成します（ルールはレンタカー会社のビジネス・ルールに従って指定されます）。

この章の内容は次のとおりです。

- Rule Author ユーザーの作成
- Rule Author の起動
- Rule Author のホーム・ページ
- レンタカー・サンプルに対するディクショナリの作成および保存
- レンタカー・サンプルに対するデータ・モデルの定義
- レンタカー・サンプルに対するビジネス用語の定義
- レンタカー・サンプルに対するルールの定義
- レンタカー・サンプルに対するルールのカスタマイズ
- Oracle Business Rules を使用した Java アプリケーションの作成
- テスト・プログラムを使用したレンタカー・サンプルの実行

注意： この章で作成するサンプル・アプリケーションを Java 版 How-To と呼びます。このサンプル・アプリケーションは、Oracle Technology Web サイトの「Oracle Business Rules」領域の「Viewlets and Tutorials」見出しの下で入手できます。

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

2.1 Rule Author ユーザーの作成

Oracle Application Server を使用している場合は、最初に適切な権限を持つユーザーを作成してから、Rule Author を起動して使用する必要があります。手順は次のとおりです。

> **注意：** これらの説明は、コンテナが JAZN XML プロバイダとともに構成されていることを前提にしています。この前提と異なる場合は、ユーザーの作成に関する情報について適切なセキュリティ・ドキュメントを参照してください。

1. Application Server Control を使用して、Rule Author がデプロイされている OC4J インスタンスに移動します。
2. **「管理」**タブをクリックします。
3. 「タスク名」列で、「セキュリティ・プロバイダ」タスクを検索し、対応する行の「タスクに移動」アイコンをクリックします。
4. **「インスタンス・レベルのセキュリティ」**をクリックします。
5. **「レルム」**タブをクリックします。
6. 「結果」セクションの表で、「ユーザー」列の番号をクリックしてユーザーを追加します。
7. **「作成」**ボタンをクリックします。

   「名前」フィールドに Rule Author へのログインに使用する名前（`ruleadmin` など）を入力します。このユーザーのパスワードを入力および確認します。「ロールの割当て」セクションで、ダブルクリックまたは矢印を使用して、このユーザーに `rule-administrators` ロールを割り当てます。作業が終了した後、**「OK」**をクリックします。

   > **注意：** Rule Author 認証が機能するには、Rule Author ユーザーが `rule-administrators` ロールに属している必要があります。

8. Rule Author アプリケーションを再起動します。

2.2 Rule Author の起動

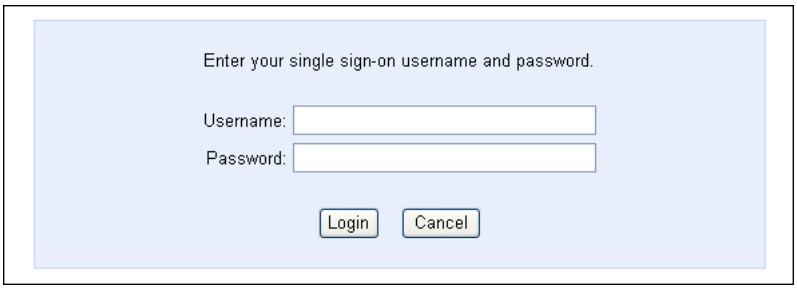
Rule Author は、ホーム・ページの URL を入力することで起動します。ホーム・ページの URL には、通常、ホスト・コンピュータ名とインストール時にアプリケーション・サーバーに対して割り当てられたポート番号、および Rule Author のホーム・ページのパスが含まれています。次に例を示します。

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

> **注意：** Oracle Application Server に対して割り当てられるポート番号は、*$ORACLE_HOME*/install ディレクトリにある readme.txt ファイルに記載されています。

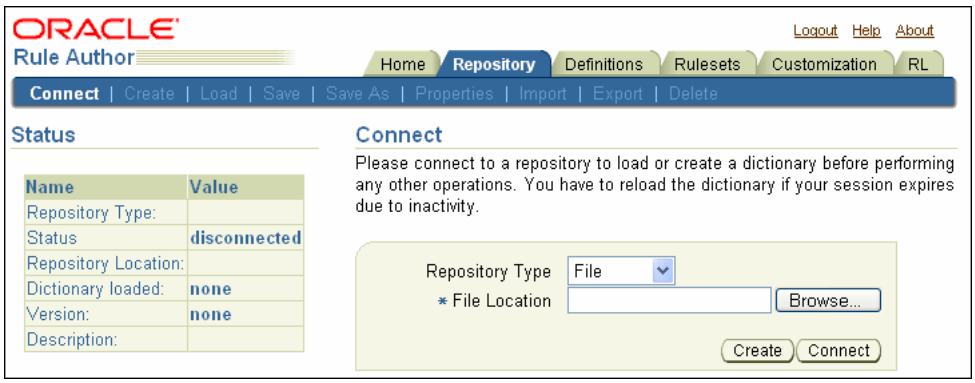
図 2-1 に、Rule Author の「ログイン」ページを示します。Rule Author ユーザーの作成（2.1 項「Rule Author ユーザーの作成」）時に指定したユーザー名とパスワードを指定します。

図 2-1 Rule Author の Single Sign-on の「ログイン」ページ



ログインしてから、「リポジトリ」タブをクリックすると、Rule Author のリポジトリ接続ページが表示されます（図 2-2 を参照）。操作を実行する前に、リポジトリに接続する必要があります。詳細は、2.4.1 項「Rule Author リポジトリへの接続」を参照してください。

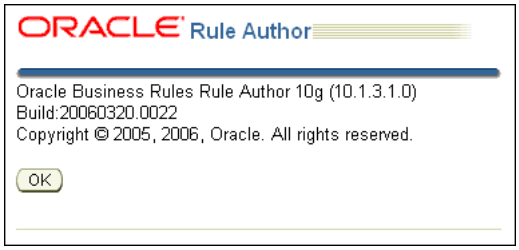
図 2-2 Rule Author の「リポジトリ」タブに表示される最初の「接続」ページ



Rule Author の各ページには、「ログアウト」、「ヘルプ」、および「バージョン情報」へのリンクが示されます。これらのリンクは次のように使用できます。

- 「ログアウト」をクリックして、「ログアウト確認」ページに移動します。このページでは、**「ログアウト」**をクリックして Rule Author をログアウトします。あるいは**「保存してログアウト」**をクリックし、変更内容を保存して Rule Author をログアウトします。ログアウトした後は、Rule Author に再度ログインする必要があります（図 2-1 を参照）。
- 「ヘルプ」をクリックすると、Rule Author のオンライン・ヘルプにアクセスします。
- 「バージョン情報」をクリックすると、Rule Author に関するバージョンおよびビルド情報が表示されます（図 2-3 を参照）。**「OK」**をクリックしてウィンドウを閉じます。

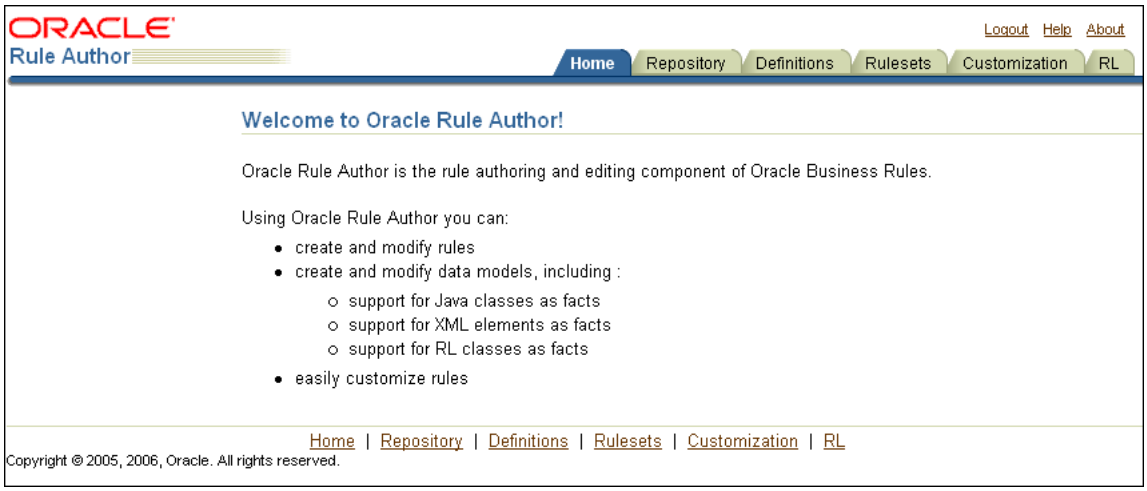
図 2-3 Rule Author の「バージョン情報」



2.3 Rule Author のホーム・ページ

「ホーム」タブをクリックして、Rule Author のホーム・ページにアクセスします（図 2-4 を参照）。ホーム・ページには 2 つのペインがあります。上部のペインには複数のタブが表示され、下部のペインには現在選択されているタブの内容が表示されます。

図 2-4 Rule Author のホーム・ページ



2.4 レンタカー・サンプルに対するディクショナリの作成および保存

Rule Author での作業は、ディクショナリから着手する必要があります。Rule Author では、ルールおよび関連する定義をディクショナリに格納します。ディクショナリを作成または保存するには、最初に、ディクショナリを格納するリポジトリに接続する必要があります。出荷時の Rule Author では、WebDAV（Web Distributed Authoring and Versioning）リポジトリとファイル・リポジトリの 2 種類のリポジトリがサポートされています。ここでは、Java 版 How-To のディクショナリを作成し、保存します。

この章の例では、ディクショナリを WebDAV リポジトリに保存します。

注意： この章に記載されているディクショナリを作成するには、WebDAV リポジトリまたはファイル・リポジトリを使用して新しいディクショナリを作成するか、あるいは Java 版 How-To とともに提供される `/dict` ディレクトリの `CarRepository` ファイル・リポジトリから完全なディクショナリをロードできます。

手順については、2.4.2 項「Rule Author ディクショナリの作成」を参照してください。

2.4.1 Rule Author リポジトリへの接続

Oracle Business Rules では、ルールおよびそのルールに関連付けられているデータ・モデルをディクショナリに格納します。ディクショナリはリポジトリに作成および保存されます。

注意： WebDAV リポジトリを使用する場合、このリポジトリが、接続する前に存在している必要があります。詳細は、付録 B「リポジトリを利用した Rule Author と Rules SDK の使用」を参照してください。

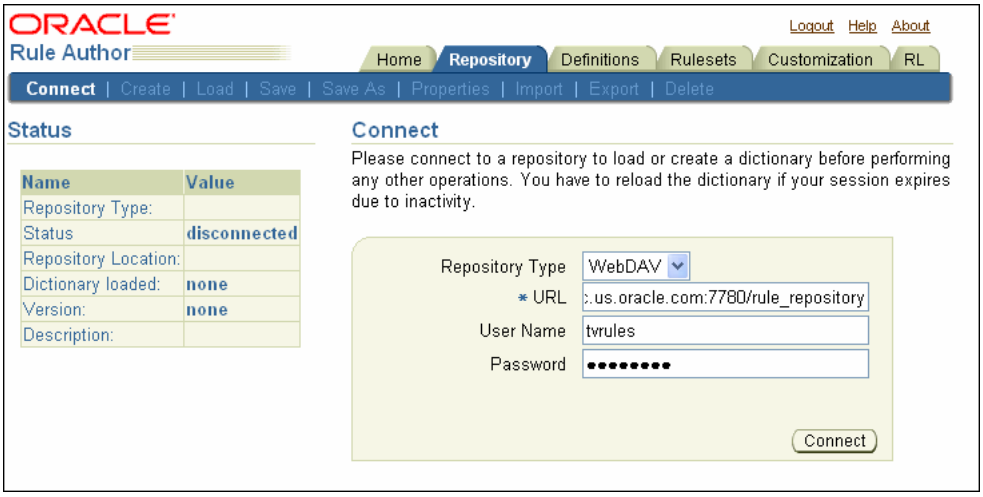
リポジトリに接続する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「接続」**第 2 タブをクリックします。
3. **「リポジトリ・タイプ」**フィールドで**「WebDAV」**リポジトリ・タイプを選択します。
4. URL を WebDAV リポジトリに入力します（図 2-5 を参照）。URL は次の形式で指定する必要があります。

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

注意： 認証が実行されるように、URL には完全修飾ホスト名を使用する必要があります。

図 2-5 Rule Author の WebDAV リポジトリの「接続」ページ



WebDAV リポジトリの設定方法の詳細は、B.1 項「WebDAV リポジトリの使用」を参照してください。

5. Rule Author が動作するシステムと WebDAV サーバーの間にプロキシがある場合、Rule Author には**「プロキシ・ユーザー」**フィールドと**「プロキシ・パスワード」**フィールドも表示されます。
6. **「接続」**をクリックします。「接続」をクリックすると、Rule Author は、確認メッセージ（図 2-6 を参照）を表示します。

注意： ファイル・リポジトリの場合、所定の時間にリポジトリを編集できるのは、そのリポジトリに格納されているディクショナリ数に関係なく常に 1 人のユーザーのみです。WebDAV リポジトリの場合は、1 人のユーザーが複数のディクショナリを同時に編集できます。

図 2-6 Rule Author の「リポジトリ」タブにある「接続」ページの「確認」

関連項目： B-3 ページ「プロキシを使用した WebDAV リポジトリへの接続」

2.4.2 Rule Author ディクショナリの作成

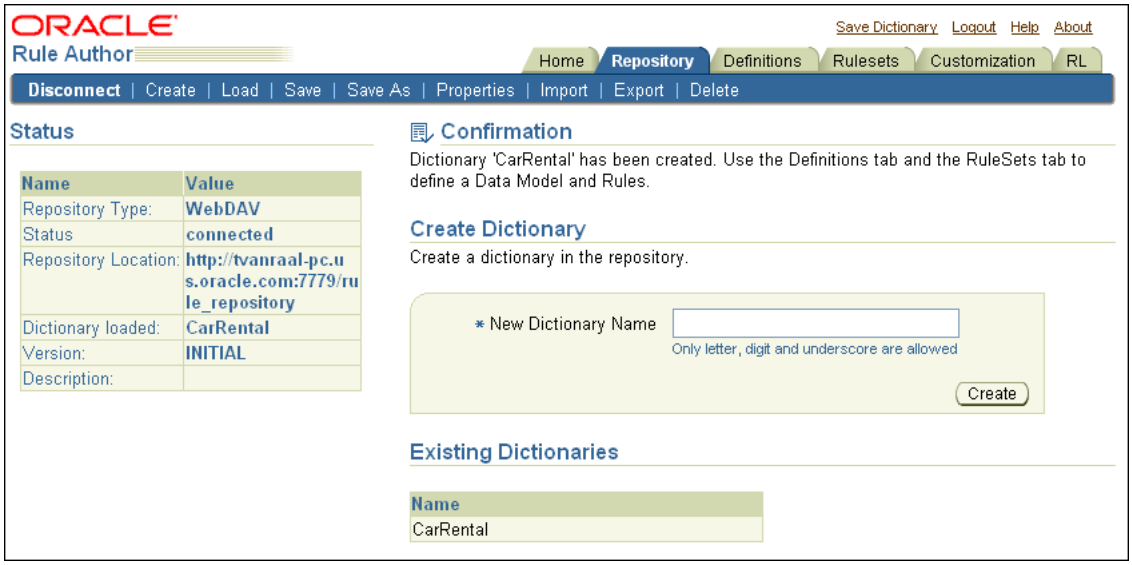
Rule Author ディクショナリは最上位コンテナであり、Rule Author で作業する起点となります。ディクショナリは、通常、アプリケーションのルール部分に対応しています。

注意： 複数のユーザーが同時に同じディクショナリを編集することは安全ではありません。

ディクショナリを作成する手順は、次のとおりです。

1. **「リポジトリ」**タブからリポジトリに接続します。
2. **「作成」**をクリックします。
3. **「新規ディクショナリ名」**フィールドにディクショナリ名を入力します。この例では、`CarRental` と入力します（図 2-7 を参照）。
4. **「作成」**をクリックします。**「作成」**をクリックすると、Rule Author は確認メッセージを表示します。

図 2-7 Rule Author の「ディクショナリを作成」ページ



注意： 独自のディクショナリの作成に加えて、Java 版 How-To とともに提供される完全なレンタカー・ディクショナリも使用できます。このディクショナリは、`/dict` ディレクトリの `CarRepository` ファイル・リポジトリにあります。このディクショナリを使用する手順は、次のとおりです。

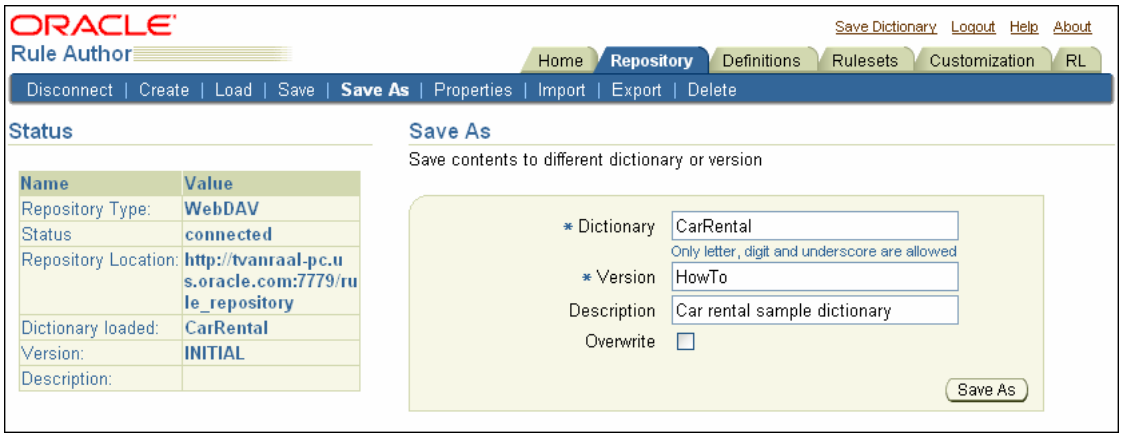
1. **「リポジトリ」**タブをクリックします。
2. **「接続」**第 2 タブをクリックします。
3. **「リポジトリ・タイプ」**ボックスで**「ファイル」**を選択します。
4. ファイル・リポジトリの完全パスを**「ファイルの場所」**フィールドに入力します。たとえば、`C:/demo/dict/CarRepository` と入力します。
5. **「接続」**をクリックします。

2.4.3 Rule Author ディクショナリのバージョン付き保存

別のディクショナリ名で保存する場合や現行のディクショナリにバージョンを指定する場合は、次のように**「別名保存」**を使用します。

1. **「リポジトリ」**タブをクリックします。
2. **「別名保存」**第 2 タブをクリックします。
3. **「ディクショナリ」**フィールドにディクショナリ名を入力します（CarRental など）。
4. バージョンを指定するには、**「バージョン」**フィールドにバージョンを入力します。たとえば、HowTo などです（図 2-8 を参照）。
5. **「別名保存」**をクリックします。**「別名保存」**をクリックすると、Rule Author によって確認メッセージが表示されます。

図 2-8 Rule Author の「別名保存」ページ



注意： Rule Author では、**「別名保存」**を使用して、同一の名前とバージョンでディクショナリを上書きできます。**「上書き」**チェック・ボックスを選択して、同じ名前とバージョンでディクショナリを保存してください。

2.4.4 Rule Author ディクショナリの保存

データの消失を防止するために、ディクショナリを定期的に保存してください。ディクショナリを保存するには、次のいずれかを実行します。

- **「リポジトリ」**タブをクリックし、次に**「保存」**第 2 タブをクリックします。
- ページ上部の**「ディクショナリの保存」**リンクをクリックします。

前述のいずれかの操作を実行した後、「ディクショナリの保存」ページの**「保存」**をクリックします。**「保存」**をクリックすると、ステータス領域に確認メッセージが表示されます。次に例を示します。

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

注意： 非アクティブ期間後は Rule Author セッションがタイムアウトになるため、作業に応じて定期的にディクショナリを保存してください。

関連項目： Rule Author のセッションのタイムアウトを構成する方法の詳細、およびタイムアウトの発生時に Rule Author が現在の作業をディクショナリ・バージョンに保存する方法の詳細は、A-3 ページの「Rule Author のセッション・タイムアウト」を参照してください。

2.5 レンタカー・サンプルに対するデータ・モデルの定義

ルールに対する作業の前に、データ・モデルを定義する必要があります。データ・モデルには、Java クラス・ファクト・タイプ、XML ファクト・タイプおよび RL Language ファクト・タイプなど、ルールで使用されるファクトやデータ・オブジェクトに関するビジネス・データ定義が格納されます。説明を単純化するために、この項では Java ファクト・タイプを Java ファクトと記述します。ここでは、Java ファクトのみを取り扱います。

内容は次のとおりです。

- レンタカー・サンプルでのファクトとしての Java オブジェクトの使用
- Rule Author への Java クラスおよびパッケージの追加
- データ・モデルへの Java クラスのインポート
- 現状の定義の保存

関連項目： 4-10 ページ「データ・モデルへの XML スキーマ要素のインポート」

2.5.1 レンタカー・サンプルでのファクトとしての Java オブジェクトの使用

Java 版 How-To には、*$HowToDir*/lib ディレクトリの car-objs.jar ファイルが格納されています。この JAR ファイルには、レンタカー・サンプルの Driver クラスが指定されています。Driver オブジェクトに対する Java ソースは、*$HowToDir*/src/carrental ディレクトリにあります。*HowToDir* には、Java 版 How-To をインストールしたディレクトリが入ります。

2.5.2 Rule Author への Java クラスおよびパッケージの追加

Java ファクトをデータ・モデルにインポートする前に、Rule Author で使用可能な Java ファクトが組み込まれているクラスおよびパッケージを作成する必要があります。そのためには、Rule Author を使用して、Java クラスが格納されているクラスパスを指定します。たとえば、Driver という Java クラスのクラスパスを追加する手順は、次のとおりです。

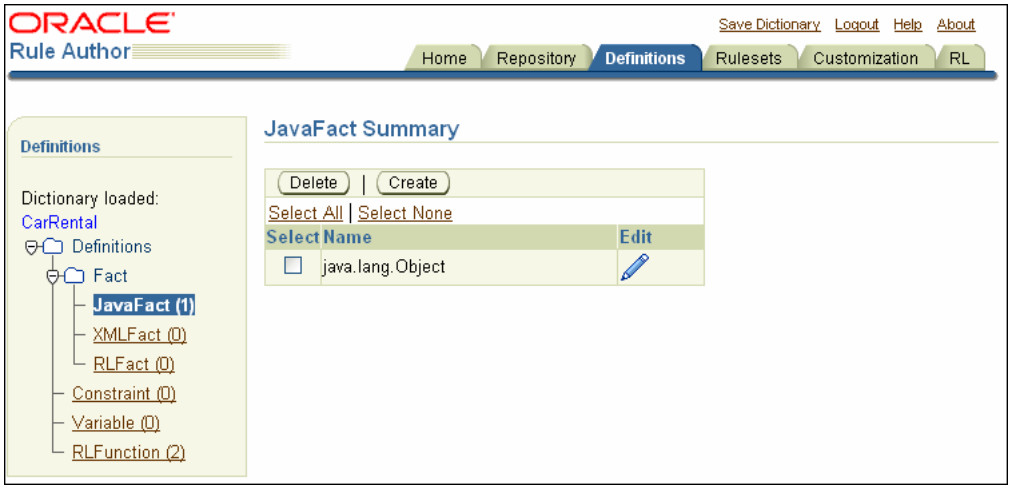
1. **「リポジトリ」**タブをクリックします。
2. 2.4.3 項でディクショナリを作成した後の場合は、手順 3 にスキップしてください。「ロード」ページで CarRental ディクショナリと HowTo バージョンを選択して、**「ロード」**をクリックします。
3. **「定義」**タブをクリックします。使用可能な定義が格納された「定義」フォルダがナビゲーション・ツリーに表示されます。メイン・ペインには、なにも表示されません。

   注意： Rule Author では、通常、下部のペインにナビゲーション・ツリーとコンテンツ領域（メイン・ペイン）が表示されます。**「定義」**タブを選択すると、ツリー上部に「定義」フォルダが表示されます。

4. ツリーの「定義」フォルダには「ファクト」フォルダが表示されます。この「ファクト」フォルダには、使用可能なファクト・タイプの **Java ファクト**、**XML ファクト**および **RL ファクト**が含まれています。

   「Java ファクト」をクリックして「Java ファクト・サマリー」ページを表示します（図 2-9 を参照）。

図 2-9 Rule Author の「定義」タブにある「Java ファクト・サマリー」ページ

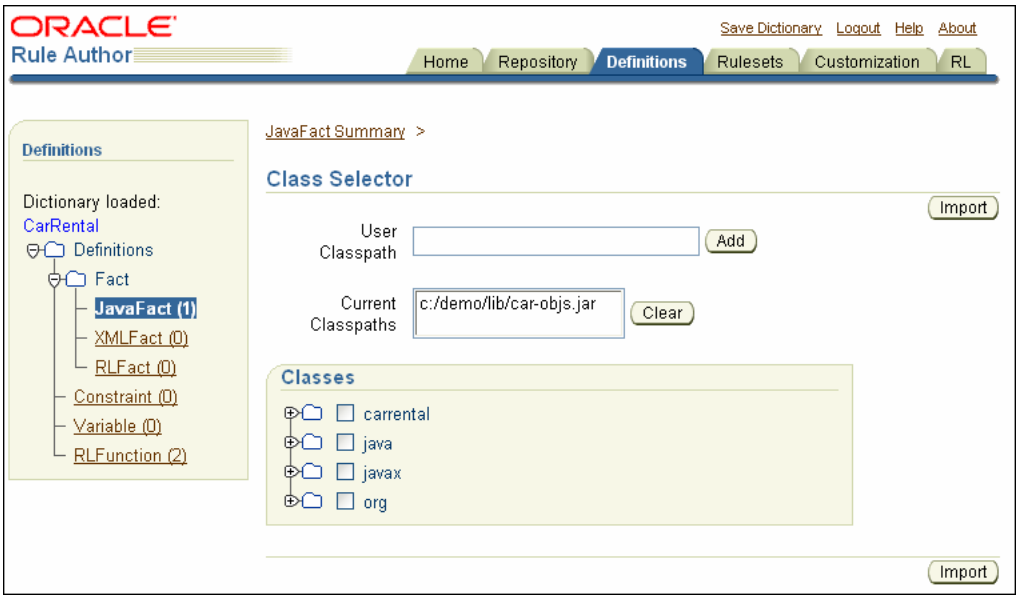


5. **「作成」**をクリックします。「クラス・セレクタ」ページが表示されます。
6. 「クラス・セレクタ」ページの**「ユーザー・クラスパス」**フィールドを使用して、クラスパスを追加できます。たとえば、Java 版 How-To では、**「ユーザー・クラスパス」**フィールドに、次のようにクラスパスを入力します。

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

   $*HowToDir* を、Java 版 How-To をインストールしたディレクトリに置換します。
7. **「追加」**をクリックします。**「現在のクラスパス」**フィールドが更新され、`carrental` パッケージが「クラス」ボックスに追加されます（図 2-10 を参照）。

図 2-10 Rule Author の「Java ファクト」の「クラス・セレクタ」ページ



関連項目： Rule Author への Java クラスおよびパッケージの追加方法については、2.5.3 項を参照してください。

2.5.3 データ・モデルへの Java クラスのインポート

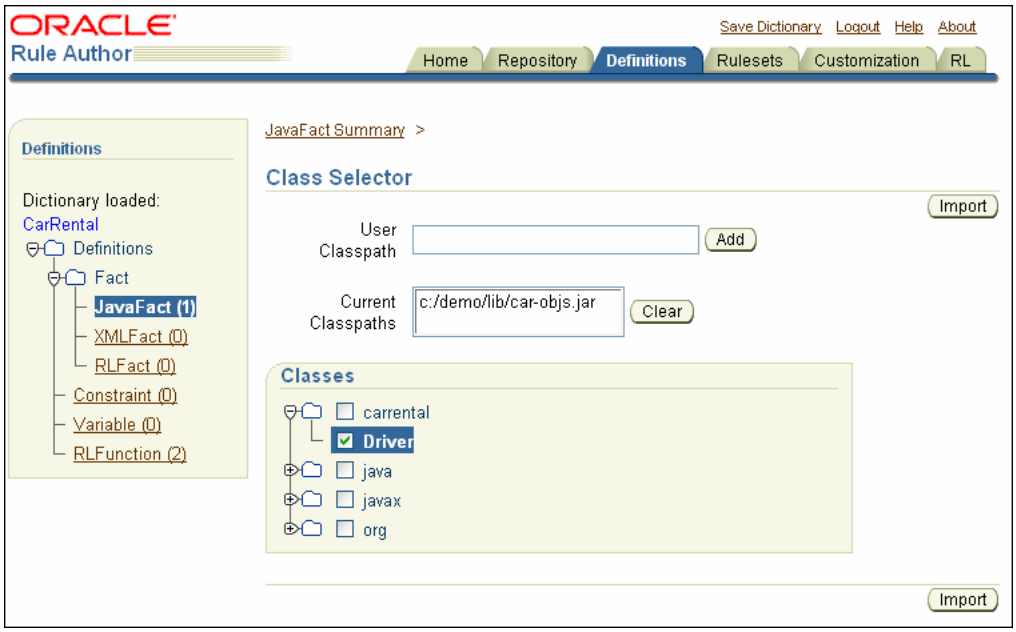
次に、「クラス・セレクタ」ページからデータ・モデルにインポートする Java クラスを選択する必要があります。Driver クラスをデータ・モデルに追加する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、「Java ファクト」フォルダをクリックします。
3. 「Java ファクト・サマリー」ページで、**「作成」**をクリックします。「クラス・セレクタ」ページが表示されます。
4. 「クラス・セレクタ」ページの「クラス」ボックスで、**carrental** ノードを開いて **Driver** チェック・ボックスを選択します（図 2-11 を参照）。
5. **「インポート」**をクリックします。「インポート」をクリックすると、Rule Author によって確認メッセージが表示されます。

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

> **注意：** クラスがインポートされると、「クラス・セレクタ」ページにインポート済クラスが太字で表示されます。

図 2-11 Rule Author の「Java ファクト」の「クラス・セレクタ」ページ



ユーザー・クラスパスを指定して、Java クラスを Rule Author にインポートする際の注意点は、次のとおりです。

- Rule Author では、Java ファクトをインポートすることは、Java の import 文と同じことを意味します。つまり、Rule Author でクラスとそのメソッドを参照できるようになります。Rule Author では、Java コードをデータ・モデルまたはディクショナリにコピーしません。
- 「クラス・セレクタ」ページには、現行のクラスパスから使用可能な Java クラスを示す「クラス」ボックスがあります。
- Rule Author のデフォルトのクラスパスには、java、javax および org の 3 つのパッケージが指定されます。これらのパッケージには、Rule Author によって Java ランタイム・ライブラリ（rt.jar）からインポートできるクラスが格納されています。Rule Author では、これらのクラスを「クラス」領域から削除できません（また、関連するクラスパスは**「現在のクラスパス」**フィールドには表示されません）。

- 同一の型ではなく、継承によって関連している 2 つのオブジェクトを割り当てたり、比較する場合は、比較する両方のクラスと継承階層内でそれらのクラス間にある全クラスをインポートする必要があります。たとえば、`ArrayList` を `Object` 型の変数に割り当てる場合は、`ArrayList`、`AbstractList` および `AbstractCollection` をデータ・モデルにインポートする必要があります。このように指定しないと、型チェックが正しく機能せず、式が有効になりません。
- Windows システムでは、円記号（¥）またはスラッシュ（/）を使用して**ユーザー・クラスパス**を指定できます。Rule Author では、どちらのパス・セパレータも使用できます。たとえば、*$HowToDir*`¥lib¥car-objs.jar` を使用できます。$*HowToDir* には、Java 版 How-To をインストールしたディレクトリが入ります。
- パフォーマンス向上のために、「クラス」ボックスのナビゲーション・ツリーは、要求に応じてレンダリングされます。したがって、子ノードは親ノードを展開した場合にのみレンダリングされます。必要なノードのみを展開することをお薦めします。
- Rule Author で使用するクラスおよびインタフェースは、次のルールに従う必要があります。

  a. クラスまたはインタフェースとそのスーパークラスを使用している場合は、最初にスーパークラスを宣言する必要があります。

  b. クラスまたはインタフェースを使用している場合、宣言できるのは、そのスーパークラスまたは実装されたインタフェースの 1 つのみです。

  詳細は、D.6 項「データ・モデル内のクラス順序と階層の保持」を参照してください。
- **「ユーザー・クラスパス」**には、JAR ファイル、ZIP ファイルまたはディレクトリのフルパスを指定できます。
- **「ユーザー・クラスパス」**にディレクトリ名を指定すると、ディレクトリは、ルート・パッケージ（完全パッケージ名の最初のパッケージ）を含んでいるディレクトリで終了するクラスパスを特定します。したがって、**「ユーザー・クラスパス」**でディレクトリを指定すると、Rule Author は、パッケージ名構造と一致するディレクトリ名をそのツリーで検索します。

  たとえば、`c:¥myprj¥cool¥example¥Test1.class` にある `cool.example.Test1` というクラスをデータ・モデルにインポートする場合は、**「ユーザー・クラスパス」**の値として `c:¥myprj` を指定します。
- Java パッケージ名には、RL Language 予約語を使用しないでください。詳細は、D.8 項「Java パッケージ名の一部としての RL 予約語の使用」を参照してください。

2.5.4 現状の定義の保存

「定義」タブでデータ・モデルを操作していて、作業を完了する場合は、ディクショナリを保存してください。

定義をディクショナリに保存する手順は、次のとおりです。

1. **「ディクショナリの保存」**リンクをクリックします。
2. 「ディクショナリの保存」ページで**「保存」**をクリックします。
3. **「定義」**タブをクリックして「定義」ページに戻ります。

2.6 レンタカー・サンプルに対するビジネス用語の定義

ビジネス用語を使用することで、ビジネス・アナリストは、Java パッケージ名、クラス名、メソッド名またはメンバー変数名を使用せずに、簡単に理解できる名前を使用して Rule Author でルールを作成できます。Rule Author には別名機能が用意されています。この機能を使用すると、ビジネスマン向けの用語を使用してルールのビジネス・オブジェクトまたはファクトを参照できます。必要な手順は、ルールに使用する予定のビジネス・オブジェクトに対して、ビジネス用語を定義することのみです。また、**「ルールセット」**タブからルールを作成する際は、Rule Author の**「参照可能」**ボックスを使用して、Rule Author のリストに表示するプロパティとメソッドを指定します。

内容は次のとおりです。

- Java ファクト定義に対するビジネス用語の指定
- 関数に対するビジネス用語の指定
- プロパティとメソッドに対する参照可能性の指定

2.6.1 Java ファクト定義に対するビジネス用語の指定

Java ファクト定義に対してビジネス用語を指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで**「Java ファクト」**ノードをクリックして「Java ファクト・サマリー」ページを表示します。レンタカー・サンプルでは、インポートされた `carrental.Driver` クラスを含む表が表示されます。
3. 編集アイコンをクリックして、`carrental.Driver` クラスに対する Java ファクトのプロパティとメソッドの表を表示します。
4. ページ上部の**「名前」**フィールドの「エイリアス」ボックスに、クラス `carrental.Driver` の**「エイリアス」**フィールドの別名 `DriverData` を入力します。
5. 「プロパティ」表の age エントリに対して、適切な別名を指定します。たとえば、**「エイリアス」**フィールドに `DriverAge` と入力します。
6. 「プロパティ」表の name エントリに対して、適切な別名を指定します。たとえば、**「エイリアス」**フィールドに `DriverName` と入力します。
7. **「OK」**をクリックし、変更内容を保存して「Java ファクト・サマリー」ページに戻ります。

注意： 変更後は必ず**「OK」**または**「適用」**をクリックしてください。**「OK」**または**「適用」**ボタンをクリックしないと、Rule Author で変更内容が保存されません。

関連項目： 3-9 ページ「データ・モデル内の Java オブジェクトの表示」

2.6.2 関数に対するビジネス用語の指定

RL Language 関数に対してビジネス用語を指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、「定義」フォルダの「RL 関数」をクリックし、「RL 関数サマリー」ページを表示します。レンタカー・サンプルでは、`DM.assertXPath` 関数および `DM.println` 関数を含む表が表示されます。
3. `DM.println` 関数について、編集アイコンをクリックして詳細を表示します。
4. **「名前」**フィールドの下にある**「エイリアス」**フィールドに別名を入力します。たとえば、**「エイリアス」**フィールドに `PrintOutput` と入力します。

> **注意：** 「エイリアス」フィールドは「関数の引数」表にもあります。この例では、「関数の引数」領域の「エイリアス」フィールドは変更しません。

5. **「OK」**をクリックし、変更内容を保存して「RL 関数サマリー」ページに戻ります。
6. ディクショナリを保存します。

2.6.3 プロパティとメソッドに対する参照可能性の指定

プロパティまたはメソッドが Rule Author のリストで参照可能かどうかを指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで**「Java ファクト」**ノードをクリックして「Java ファクト・サマリー」ページを表示します。レンタカー・サンプルでは、インポートされた `carrental.Driver` クラスを含む表が表示されます。
3. 編集アイコンをクリックして、`carrental.Driver` に対する Java ファクトのプロパティとメソッドの表を表示します。
4. 「プロパティ」表の各エントリについて、「参照可能」チェック・ボックスを使用して適切な参照可能性を指定します。この例では、メンバー変数 `age` および `name` のみを参照可能にする必要があります。
5. **「OK」**をクリックし、変更内容を保存して「Java ファクト・サマリー」ページに戻ります。
6. ディクショナリを保存します。

> **注意：** 特定のプロパティまたはメソッドに対する参照可能インジケータを変更すると、依存する定義またはルールが誤って表示されることがあります。この状況が発生する場合は、問題の原因となっている参照不可のプロパティまたはメソッドを参照可能に設定してください。

2.7 レンタカー・サンプルに対するルールの定義

この項では、レンタカー・サンプルのルールを定義し、Rule Author でルールを作成する基本的な手順を示します。

内容は次のとおりです。

- レンタカー・サンプルに対するルールセットの作成
- レンタカー・サンプルに対するルールの作成

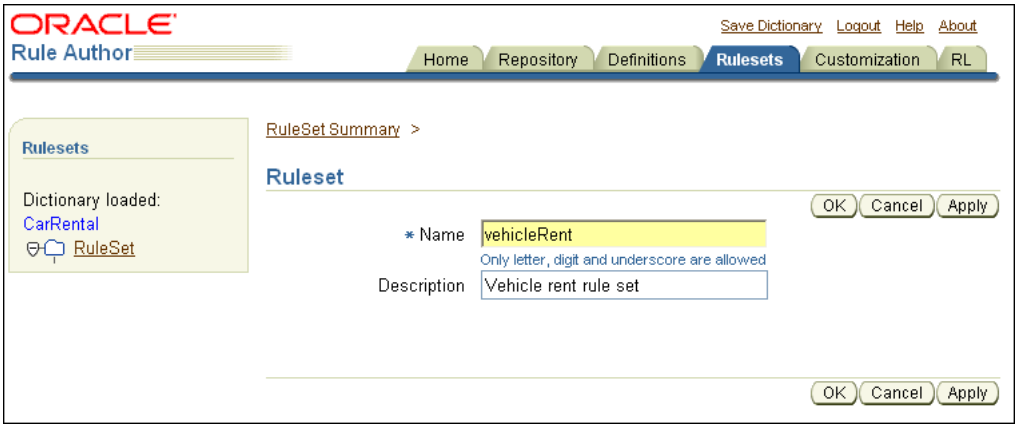
2.7.1 レンタカー・サンプルに対するルールセットの作成

Rule Author を使用してルールを作成する前に、ルールセットを作成する必要があります。ルールセットとはルールのコンテナです。

ルールセットを作成する手順は、次のとおりです。

1. **「ルールセット」**タブをクリックします。
2. ナビゲーション・ツリーで**「ルールセット」**ノードをクリックします。
3. 「ルールセット・サマリー」ページで、**「作成」**をクリックします。「ルールセット」ページが表示されます。
4. **「名前」**フィールドにテキストを入力します。たとえば、**「名前」**フィールドに `vehicleRent` と入力します。
5. （オプション）**「説明」**フィールドにルールセットの説明を入力します（図 2-12 を参照）。

図 2-12 Rule Author の「ルールセット」ページ



6. **「OK」**をクリックして、`vehicleRent` ルールセットを作成します。**「OK」**をクリックすると、ナビゲーション・ツリーの「ルールセット」の下に新しいルールセットが表示されます。
7. ディクショナリを保存します。

注意： ルールセットを削除する手順は、次のとおりです。

1. ナビゲーション・ペインで「ルールセット」フォルダを選択します。
2. **「選択」**フィールドのチェック・ボックスを選択することで、「ルールセット」領域で適切なルールセットを選択します。
3. **「削除」**を選択します。

2.7.2 レンタカー・サンプルに対するルールの作成

ルールセットを作成した後は、ルールセットにルールを作成できます。ここでは、UnderAge というルールを作成します。このルールは、次の内容をテストします。

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

UnderAge ルールには、Rules Engine が照合する単一のパターンと、そのパターンに適用されるテストが指定されます。

次のアクションが UnderAge ルールに関連付けられます。

- テキスト「Rental declined」、一致した運転者の名前、メッセージ「Under Age, age is: 」、および運転者の年齢の出力。
- 一致した運転者オブジェクトの取消し。様々な理由から、ファクトの取消しが必要になることがあります。たとえば、ファクトの使用が終了して、Rules Engine から削除する場合や、ルールに関連付けられているアクションによって状態が変化したため、Rules Engine の現在の状態が示されるように、ファクトを取り消す必要がある場合などです。

2.7.2.1 レンタカー・サンプルに対する UnderAge ルールの追加

Rule Author を使用して UnderAge ルールを追加する手順は、次のとおりです。

1. **「ルールセット」**タブをクリックします。ナビゲーション・ペインには、2-15 ページの「レンタカー・サンプルに対するルールセットの作成」の項で作成した `vehicleRent` ルールセットが格納された「ルールセット」フォルダが表示されます。
2. ナビゲーション・ツリーで、vehicleRent フォルダをクリックします。「ルールセット」ページが表示され、ルールが表にリストされます（図 2-13 を参照）。

> **注意：** 以前に作成したルールがない場合、「ルール」表は空です。

図 2-13 Rule Author の「ルールセット」ページに表示されている「ルール」表

3. **「作成」**をクリックします。「ルール」ページが表示されます。
4. **「名前」**フィールドに `UnderAge` と入力します。

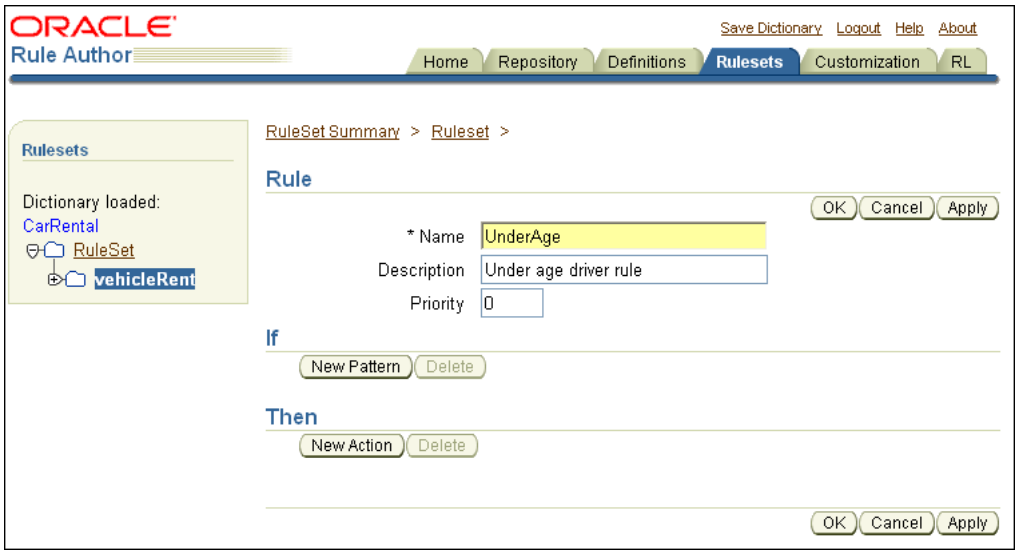
5. **「優先度」**フィールドのデフォルト値（0）は変更しないでください。

注意： **「優先度」**フィールドでルールの優先度が決まります。ルール優先度は、ルールセットの複数のルールが適用される場合に機能するルールとその順序を指定します。ルールを使用するアプリケーションの多くは、決定が下されるまではルールセットのルールが順不同で適用されるため、ルール優先度の設定は必須ではありません。

---

6. **「説明」**フィールドに `Under age driver rule` と入力します（図 2-14 を参照）。

図 2-14 Rule Author の「ルール」ページ



関連項目： ルール優先度の使用に関する詳細は、『Oracle Business Rules ランゲージ・リファレンス』を参照してください。

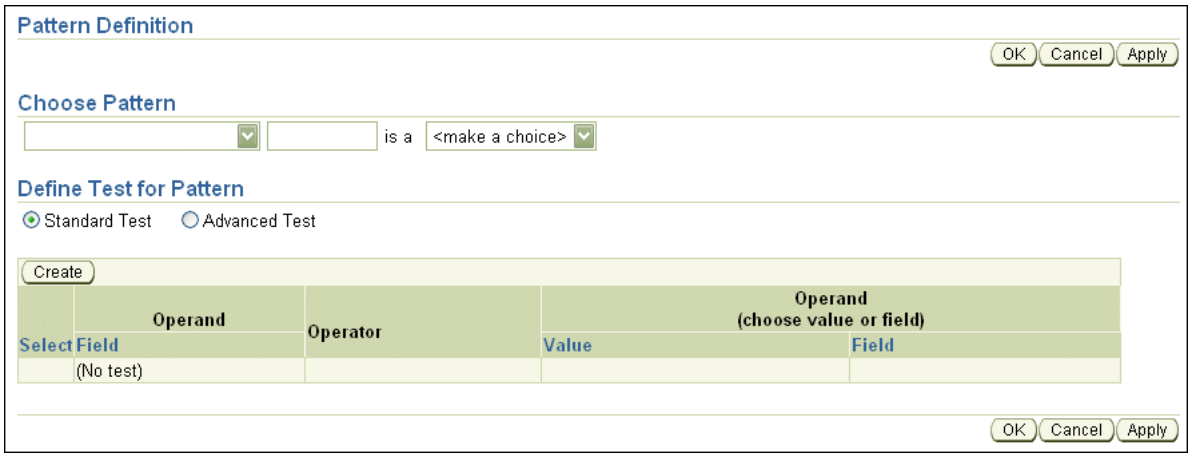
2.7.2.2 UnderAge ルールへのパターンの追加

Rules Engine を実行すると、ルール・パターンに対してファクトがチェックされ、一致するパターンが検索されます。UnderAge ルールのパターンを追加するには、次の手順を実行します。

1. 「ルール」ページで、**「if」**ボックスから**「新規パターン」**をクリックします。「パターン定義」ページが表示されます。このページには、「パターンの選択」と「パターンのテストを定義」の 2 つの領域があります（図 2-15 を参照）。

注意： 「パターン定義」ページが表示されない場合は、ブラウザのポップアップ遮断機能が使用可能になっている可能性があります。Rule Author を使用するには、ポップアップ遮断機能を使用不可にする必要があります。

図 2-15 Rule Author の「パターン定義」ページ



2. **「パターンの選択」**の下にある最初のボックスで、最初の選択肢を選択します（選択ボックスに値は表示されません）。

   このボックスは、一致するたびに（一致するすべての運転者に）、ルールを起動することを指定します。別の選択肢の 1 つである**「次のケースが少なくとも 1 つある」**の値は、少なくとも 1 つが一致する（そのような運転者が 1 人いる）場合に、ルールを 1 回行うことを選択します。**「次のケースがない」**の値は、一致しない（一致する運転者がいない）場合に、ルールを 1 回起動することを指定します。

3. **「パターンの選択」**の下にある隣の領域には、一致したファクトに暫定的な名前を入力できます。このフィールドに `driver` と入力（パターンのバインド変数名を定義）します。

   この値を使用して、単一のルールで同じタイプの複数のインスタンスをテストできます。たとえば、パターンのバインド変数を使用すると、`driver1.age > driver2.age` などの比較で、指定した名前を使用して、ある運転者と他の運転者との一致を比較できます。

4. 3 番目のボックスには、`<make a choice>` というテキストが表示されます。このボックスには、データ・モデルから使用可能なファクト・タイプが表示されます。このボックスで、`DriverData`（ファクト・タイプの別名をデータ・モデルに定義していない場合は `carrental.Driver`）を選択します。

5. **「OK」**をクリックし、パターン定義を保存して「ルール」ページに戻ります。

6. 「ルール」ページで**「OK」**をクリックし、ルールを保存します。

注意： パターンの変更内容は、「ルール」ページで**「OK」**または**「適用」**をクリックするまで、ルールには追加されません。**「OK」**または**「適用」**をクリックせずに別のルールセットにナビゲートしたり、異なるタブを選択すると、パターン定義は破棄されます。

---

7. ディクショナリを保存します。

パターンに対してテストが定義されていないと、定義するアクションはすべての運転者に適用されます。パターンにテストを定義するには、2.7.2.3 項の説明に従って「パターン定義」ページを続行します。

関連項目： 2-21 ページ「UnderAge ルールに対するアクションの追加」

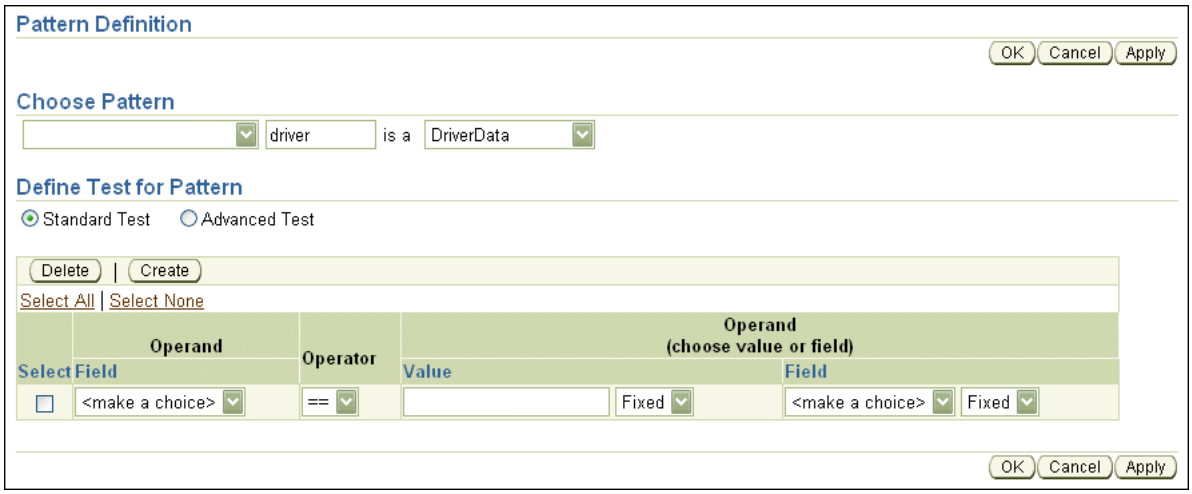
2.7.2.3 UnderAge ルールでのパターンに対するテストの定義

パターンに対してテストを追加する手順は、次のとおりです。

1. **「ルールセット」**タブのナビゲーション・ペインで、テストを追加するルールをクリックします。この例では、ナビゲーション・ペインで UnderAge ルールのノードをクリックします。「ルール」ページが表示されます。
2. 「ルール」ページの**「if」**ボックスで、鉛筆アイコンをクリックして「パターン定義」ページを表示します。「パターン定義」ページには、**「パターンの選択」**と**「パターンのテストを定義」**の 2 つの領域があります。

3. 「パターン定義」ページで、**「標準テスト」**ボタンを選択し、次に**「作成」**をクリックします（図 2-16 を参照）。詳細は、3.8.1 項「「アドバンスト・テスト式」オプションの使用」を参照してください。

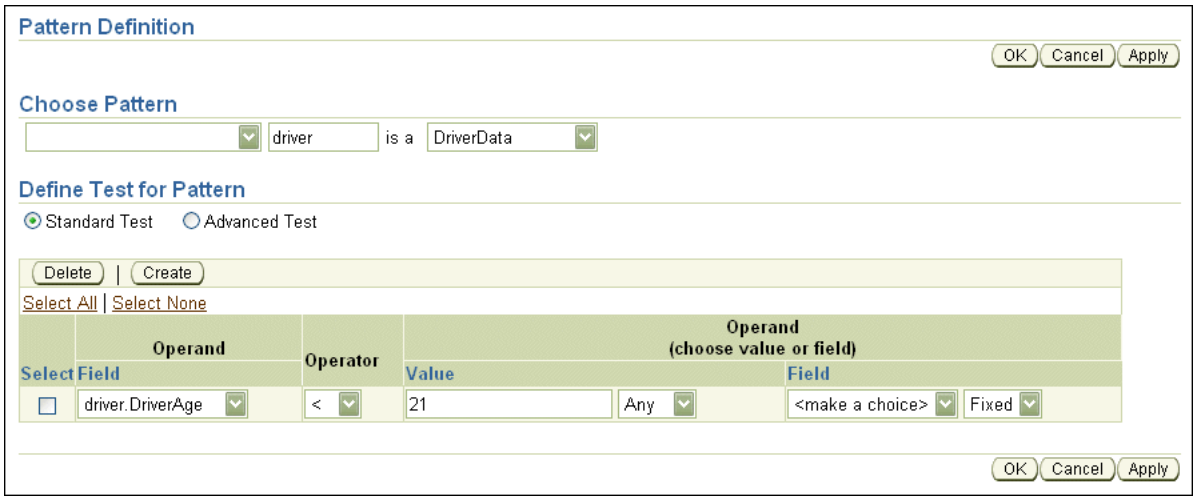
図 2-16 Rule Author のルールの「パターン定義」ページの「パターンのテストを定義」フィールド



4. **「オペランド」**列で、最初の**「フィールド」**ボックスから `driver.DriverAge`（このメンバー変数の別名をデータ・モデルに定義していない場合は `driver.age`）を選択します。
5. **「演算子」**列で、<（より小さい）を選択します。
6. **「オペランド」**列で、**「値」**ボックスに 21 を入力します。**「フィールド」**ボックスには値を入力しません。
7. **「値」**ボックスと**「フィールド」**ボックスの横には、`Any` と `Fixed` の固定値を備えたドロップダウン・リストがあります（図 2-17 を参照）。これらの値は制約と呼ばれ、カスタマイズを有効化または無効化します。`Fixed` を使用すると、フィールドは読取り専用になります。これにより、このフィールドはカスタマイズなしに指定されます。`Any` を選択すると、Rule Author で値を変更できることが指定されます（これにより、技術を持たないユーザーでも、「カスタマイズ」タブからフィールドに変更を加えることができます。）。

   「値」フィールドの制約として `Any` を選択します。

図 2-17 UnderAge ルールの値が表示された Rule Author の「パターン定義」ページ



8. **「OK」**をクリックし、変更内容を保存して「ルール」ページに戻ります。
9. 「ルール」ページで**「OK」**をクリックします。

注意： パターンの変更内容は、「ルール」ページで**「OK」**または**「適用」**をクリックするまで、ルールには追加されません。**「OK」**または**「適用」**をクリックせずにルールセットにナビゲートしたり、異なるタブを選択すると、パターン定義は破棄されます。

---

10. ディクショナリを保存します。

パターンに対してテストを追加する際の注意点は、次のとおりです。

- **標準テスト**式では、テストが true と評価されるのは、定義するすべてのテストが一致した場合のみです。また、標準テストでは、グループ化や、パラメータを使用する関数は許可されません。ただし、標準テストを使用すると、カスタマイズに対して制約を定義できます。
- **アドバンスト・テスト**式では、テストに標準テストのような制限はありませんが、制約は使用できません。アドバンスト・テスト式は、RL Language を使用して直接保存されることはありません。Rule Author によって式に別名が取り込まれるためです。RL Language では、別名はサポートされていません（Rule Author で別名は変数名にマップされます）。

関連項目：

- 2-24 ページ「レンタカー・サンプルに対するルールのカスタマイズ」
- 3-3 ページ「制約の使用」

2.7.2.4 UnderAge ルールに対するアクションの追加

アクションはパターン一致と関連しています。ルールの「if」部が一致すると、Rules Engine によって「then」部がアクティブになり、ルールに関連するアクションの実行が準備されます。

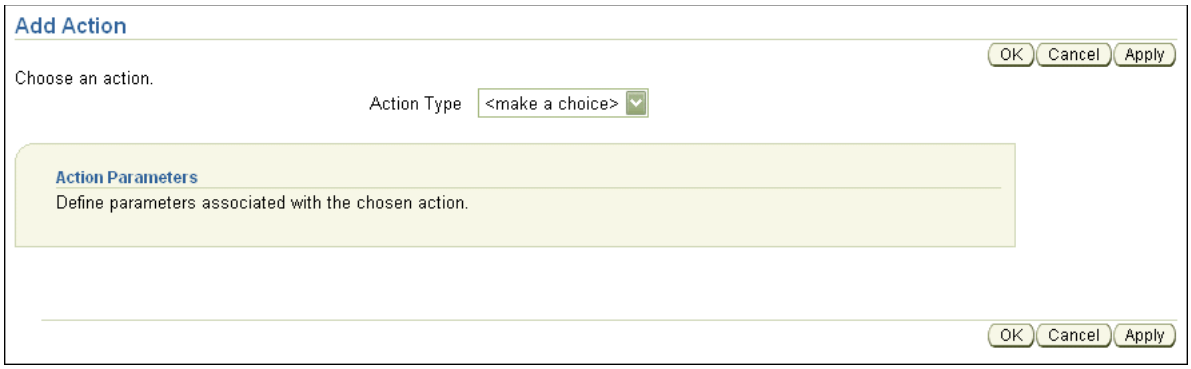
ここでは、UnderAge ルールに対して 2 つのアクションを追加します。第 1 のアクションは結果の出力です。第 2 のアクションは、Rules Engine からの運転者ファクトの取消しです。次のような様々な理由で、ファクトの取消しが必要になることがあります。

- ファクトの使用が終了し、Rules Engine から削除する場合。
- ルールに関連付けられているアクションによって状態が変化したため、Rules Engine の現在の状態が示されるように、ファクトを取り消す必要がある場合。

UnderAge ルールの一致に関する結果を出力するアクションを追加する手順は、次のとおりです。

1. **「ルールセット」**タブをクリックします。
2. ツリーで、vehicleRent フォルダの下の UnderAge ルールのノードをクリックします。
3. 「ルール」ページで、**「then」**セクションの**「新規アクション」**をクリックします。「アクションの追加」ページが表示されます（図 2-18 を参照）。

図 2-18 Rule Author の「アクションの追加」ページ



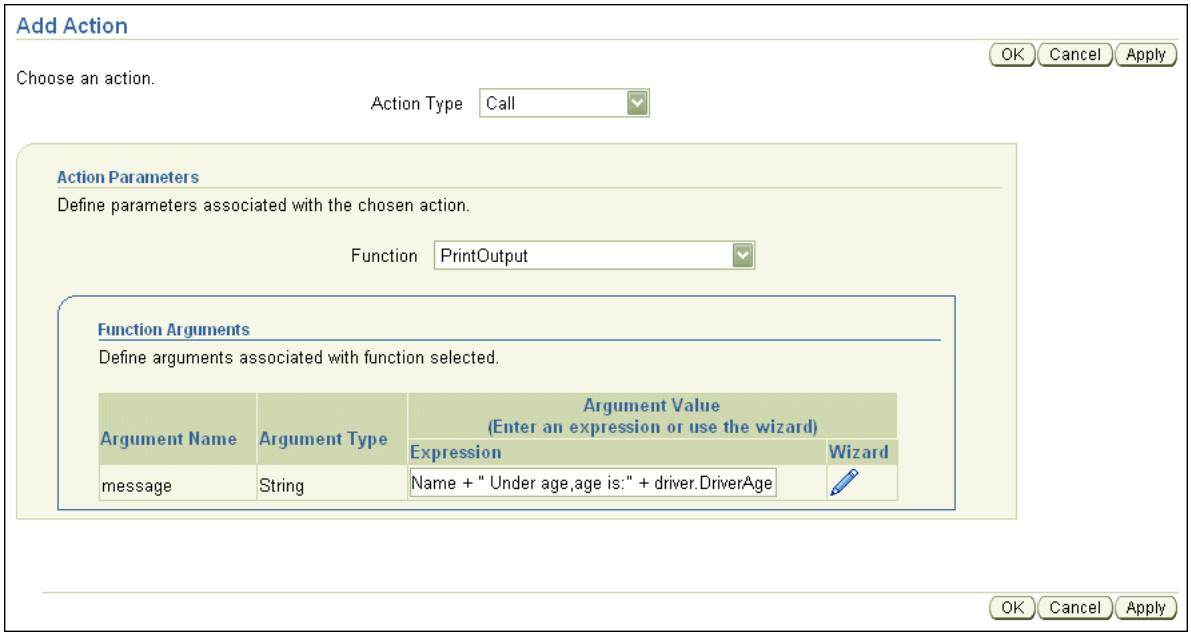
4. **「アクション・タイプ」**ボックスから、Call 項目を選択します。「アクション・パラメータ」ボックスが表示されます。
5. **「関数」**リストから PrintOutput（この関数の別名を定義していない場合は DM.println）を選択します。「関数の引数」ボックスが開きます。
6. 「関数の引数」ボックスの**「引数の値」**フィールドの**「式」**列に、値を入力します（図 2-19 を参照）。次に例を示します。

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

注意 1: Rule Author では、Java に類似した構文を式に対して使用します。RL Language は完全な式構文を定義します。

注意 2: 式に使用される変数名が不明な場合は、「ウィザード」フィールドで編集アイコンを使用して、式ウィザードを起動します。式を記述するための追加領域を備えたウィザード・ページが表示されます。このウィザード・ページには変数選択ボックスもあり、変数をより簡単かつ正確に入力することができます。

図 2-19 UnderAge ルールに関する Rule Author の「アクションの追加」ページ



7. **「OK」**をクリックし、変更内容を保存して「ルール」ページに戻ります。
8. 「ルール」ページで、**「OK」**をクリックします。
9. ディクショナリを保存します。

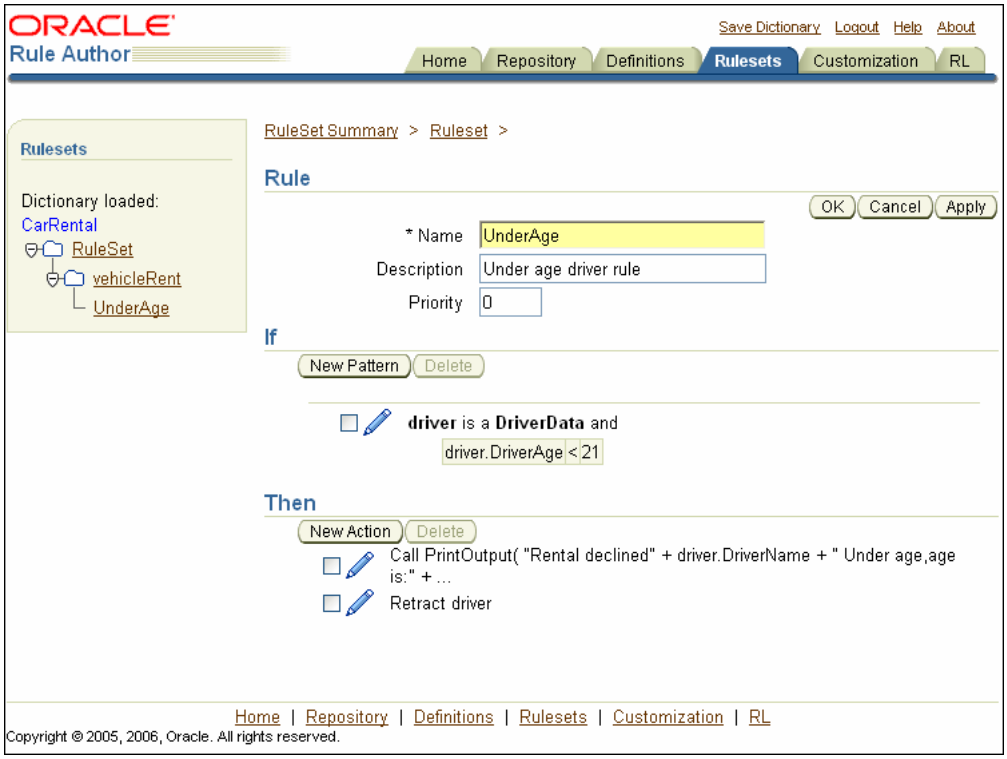
次に、UnderAge ルールに対して取消しアクションを追加します。次の手順を実行し、この第 2 のアクションをルールに追加します。

1. **「ルールセット」**タブをクリックします。
2. vehicleRent フォルダの下の UnderAge ルールのノードをクリックします。
3. 「ルール」ページで、**「then」**ボックスから**「新規アクション」**をクリックします。「アクションの追加」ページが表示されます。
4. **「アクション・タイプ」**ボックスから `Retract` を選択します。**「アクション・パラメータ」**ボックスが表示されます。
5. 「アクション・パラメータ」領域の**「ファクト・インスタンス」**リストから、`driver` を選択します。パターン名 `driver` は、パターンが一致した単一のインスタンスを参照します。
6. **「OK」**をクリックし、変更内容を保存して「ルール」ページに戻ります。
7. 「ルール」ページで、**「OK」**をクリックします（図 2-20 を参照）。

> **注意：** アクションの変更内容は、「ルール」ページで**「OK」**または**「適用」**をクリックするまで、ルールに追加されません。**「OK」**または**「適用」**をクリックせずにルールセットにナビゲートしたり、異なるタブを選択すると、Rule Author によってアクション定義は破棄されます。

8. ディクショナリを保存します。

図 2-20 Rule Author の UnderAge ルールのパターンとアクション



注意： アクションをルールに追加する場合、連続して追加できるのは新しいアクションのみです。あるアクションがそれ以前のアクションの結果に依存する場合は、アクションを追加する順序が重要になります。アクション・タイプの詳細は、表 3-2 を参照してください。

関連項目：『Oracle Business Rules ランゲージ・リファレンス』

2.8 レンタカー・サンプルに対するルールのカスタマイズ

Rule Author の**「カスタマイズ」**タブは、ビジネス・ユーザー用に設計されています。ルール開発者は、**「ルールセット」**タブから使用できる「パターン定義」ページの**許可された値**のフィールドを使用して、カスタマイズを許可するかどうかを指定します。カスタマイズが許可されている場合は、カスタマイズ可能なフィールドについて、有効な値の範囲を指定できます。その結果、ビジネス・ユーザーは**「カスタマイズ」**タブを使用して値を変更できます。

この例では、**「カスタマイズ」**タブで、運転者の年齢制限を変更するように UnderAge ルールを変更できます（このサンプルでは、値に制限はなく、あらゆる指定値が有効です）。

「カスタマイズ」タブを使用して UnderAge ルールを変更する手順は、次のとおりです。

1. **「カスタマイズ」**タブをクリックします。ナビゲーション・ペインには vehicleRent フォルダが表示され、その下に、後ろにアスタリスク（*）の付いた UnderAge ルールのノードが表示されています。このアスタリスクは、ルールがカスタマイズ可能なことを示しています。
2. UnderAge ルールのノードをクリックします（図 2-21 を参照）。

図 2-21 UnderAge ルールに関する Rule Author の「ルールのカスタマイズ」ページ

3. 「ルールのカスタマイズ」ページの「パターンのカスタマイズ」ボックスには、`driver.DriverAge < 21` のテストを編集できるテキスト入力フィールドがあります。

   このフィールドで、値 21 を 19 に変更します。
4. **「適用」**をクリックします。
5. ディクショナリを保存します。

ディクショナリを保存すると、Java 版 How-To に対するデータ・モデルとルールの作成は終了となります。

関連項目： 許可された値のフィールドの詳細は、2-19 ページの「UnderAge ルールでのパターンに対するテストの定義」を参照してください。

2.9 Oracle Business Rules を使用した Java アプリケーションの作成

データ・モデルおよびルール指定したルールセットが格納されているディクショナリを作成および保存した後は、ルール対応 Java アプリケーションでディクショナリを使用できます。この項では、ルール対応 Java アプリケーションを作成する手順を示します。

注意： Java コールは必ず try/catch ブロックにラップされるようにしてください。

内容は次のとおりです。

- Rules SDK および Rules RL Language クラスのインポート
- Rules SDK でのリポジトリの初期化
- Rules SDK でのディクショナリのロード
- ルールセットの指定と Rules SDK での RL Language の生成
- ルール・セッションの初期化および実行
- ルール・セッション内のビジネス・オブジェクトのアサート
- ルール・セッションでの実行関数の使用

このサンプル・アプリケーションの完全なコードは、$*HowToDir*/src/carrental ディレクトリの TestMain.java を参照してください。$*HowToDir* には、Java 版 How-To をインストールしたディレクトリが入ります。

注意： この章に前述されている項の説明に従うことで、WebDAV リポジトリおよび CarRental というディクショナリが作成および保存されています。How-To サンプル・コードで提供されているレンタカーの例でも、CarRental という名前のディクショナリが格納されているファイル・リポジトリを使用します。この章で作成した WebDAV リポジトリと、How-To サンプルのファイル・リポジトリのディクショナリ内容は同じです。

How-To サンプルのコードには、WebDAV リポジトリとファイル・リポジトリの両方のコードが含まれていますが、ファイル・リポジトリのみが詳細に説明されています。How-To サンプルでは、移植性の観点からファイル・リポジトリを使用していますが、前述の項で作成した WebDAV リポジトリを使用するように変更できます。

2.9.1 Rules SDK および Rules RL Language クラスのインポート

ルール対応プログラムを記述する最初の手順は、必要なクラスのインポートです。例 2-1 に、レンタカー・サンプルのための TestMain.java アプリケーションからのインポートを示します。

例 2-1 Rules SDK でのレンタカー・サンプルに必要なインポート

```
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 Rules SDK でのリポジトリの初期化

ルール対応 Java アプリケーションを作成する場合は、次の手順を実行し、ディクショナリにアクセスしてルールセットを指定します（例 2-2 を参照）。

1. リポジトリのパスが指定されている `String` を作成します。
2. Rules SDK の `RuleType` オブジェクトを使用して、`RepositoryManager.getRegisteredRepositoryType` メソッドから取得するリポジトリを保持します。例 2-2 は、ファイル・タイプ・リポジトリを示しています。
3. リポジトリ・マネージャの `createRuleRepositoryInstance` メソッドを使用してリポジトリ・インスタンスを作成します。
4. `RepositoryContext` を定義し、適切なプロパティを設定します。ファイル・リポジトリの場合は、例 2-2 の `repoPath` パラメータに伴って記載されているように、リポジトリのパスを指定します。
5. `RuleRepository` オブジェクト `repo` の `init` メソッドを使用して、リポジトリ・インスタンスを初期化します。

例 2-2 Rules SDK でのディクショナリのロード

```
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 );
```

例 2-2 のように、ファイル・リポジトリではなく WebDAV リポジトリをロードする場合は、`getWebDAVRepository` メソッドを使用してください。コメントには、*HowToDir*`/src/carrental` ディレクトリの `TestMain.java` が含まれます。$*HowToDir* には、Java 版 How-To をインストールしたディレクトリが入ります。

2.9.3 Rules SDK でのディクショナリのロード

ルールを使用する Java アプリケーションを作成する場合は、指定されたバージョンのディクショナリをロードする必要があります。例 2-3 のように、`RuleDictionary` オブジェクトを使用してディクショナリをロードします。この例では、`CarRental` ディクショナリを `HowTo` バージョンで `dict` という名前のオブジェクトにロードします。`CarRental` ディクショナリが、リポジトリで使用可能である必要があります。バージョン名 `HowTo` の `CarRental` ディクショナリは、以前に Rule Author を使用して作成されています。

例 2-3 Rules SDK でのディクショナリのロード

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

2.9.4 ルールセットの指定と Rules SDK での RL Language の生成

ディクショナリをロードした後は、ルールセットを指定し、Rules SDK を使用して RL Language プログラムを生成する必要があります。この手順は、ディクショナリが中間 XML 形式を使用してデータ・モデルとルールを格納するために必要です。Rules SDK には、ルールにアクセスするためのメソッドと関連するデータ・モデルがあります。Rules SDK は、中間 XML 形式からのマッピングを実行して、Rules Engine で動作する RL Language プログラムを作成します。

Rule Author では、各ルールセットには 2 つのコンポーネント（ディクショナリのルールセットすべてに適用されるグローバルなデータ・モデルと、ルールセットに関連付けられている一連のルール）が挿入されます。例 2-4 に、これら 2 つのコンポーネントについて RL Language を生成するコードを示します。

例 2-4 Rules SDK での Oracle Business Rules Rule Language の生成

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

2.9.5 ルール・セッションの初期化および実行

ルールとデータ・モデルが挿入されている RL Language プログラムを生成すると、いつでもルール・セッションで作業できる状態になります。ルール・セッションは、Rules Engine を初期化し、多数のルール実行にわたって Rules Engine の状態を保守します。`RuleSession` オブジェクトは、アプリケーションと Rules Engine との間のインタフェースです。

例 2-5 に、`RuleSession` オブジェクトを作成し、RL Language プログラムを実行するコードを示します。

`executeRuleset()` メソッドは、文字列として渡された RL Language プログラムを実行します。このメソッドは、指定の RL Language プログラムの解析を Rules Engine に指示します。

注意： `executeRuleset()` コールの順序は重要です。ルールセットの RL Language プログラムを実行する前に、データ・モデルの RL Language プログラムを実行する必要があります。データ・モデルには、関連するルールセットを実行するときに必要なグローバル情報が格納されます。

例 2-5 Rules SDK でのルール・セッションの初期化および実行

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

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

データ・モデルとルールセットがロードされると、ルール・セッションは、そのルール・セッション用にアサートするファクトに対していつでも指定のルールセットを実行できる状態になります。

2.9.6 ルール・セッション内のビジネス・オブジェクトのアサート

ルール・セッションを実行する前に、通常はいくつかのファクトをアサートします。ルール・セッションでデータ・モデルを実行する場合は、新しいファクトをアサートするルール・セッションを準備します。ファクトをアサートするには、`session.callFunctionWithArgument()` メソッドと、ファクトを引数として提供する `assert` 関数を使用します。

例 2-6 に、レンタカー・サンプルの `Driver` オブジェクトを準備して、3 つのファクトをアサートするサンプル・コードを示します。

例 2-6 レンタカー・サンプルに対する運転者および事故記録の準備

```
// 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 ルール・セッションでの実行関数の使用

例 2-7 に、ルール・セッションを実行するコードを示します。

例 2-7 実行関数を使用したルール・セッションの実行

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

2.10 テスト・プログラムを使用したレンタカー・サンプルの実行

$HowToDir/lib ディレクトリには、TestMain.jar が格納されています。これは、CarRental ディクショナリを使用する Oracle Business RulesJava アプリケーションで、ただちに実行できます。ディクショナリ名を変更した場合は、TestMain.java を変更する必要があります（ソースはディレクトリ $*HowToDir*/src にあります。*HowToDir* には、Java 版 How-To をインストールしたディレクトリが入ります）。

例 2-8 に、TestMain 内でアサートされたファクトを使用して実行している TestMain からの出力を示します。

注意： $*HowToDir*/docs ディレクトリの Readme.html ファイルには、テスト・プログラムの実行に必要な環境変数の設定について指示が記載されています。

例 2-8 レンタカー・プログラムの実行サンプル

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

すべてのファクトがルールにマッチして出力を生成するわけではないことに注意してください。この例は、UnderAge ルールに一致する、アサートされたファクトである運転者に関する出力を示しています。

3

Rule Author 機能の使用

この章では、Rule Author の高度な機能の一部を使用する方法について説明します。

この章の内容は次のとおりです。

- 変数の使用
- 制約の使用
- RL ファクトの使用
- 関数の使用
- ルールの使用
- データ・モデル内の Java オブジェクトの表示
- Oracle Business Rules RL Language テキストの生成
- Rule Author のディクショナリ・プロパティの構成
- Rule Author のディクショナリの削除
- ディクショナリのインポートとエクスポート
- テスト・ルールセットの使用
- ルールの起動と Rules Engine 結果の取得

3.1 変数の使用

この項では、Rule Author を使用して、メッセージの一部を置換する変数を追加します。このメッセージは、第 2 章で作成した Java 版 How-To で出力するメッセージです。Oracle Business Rules で使用する変数は、Java におけるパブリックな静的変数に類似しています。変数は、定数または変更可能に指定できます。

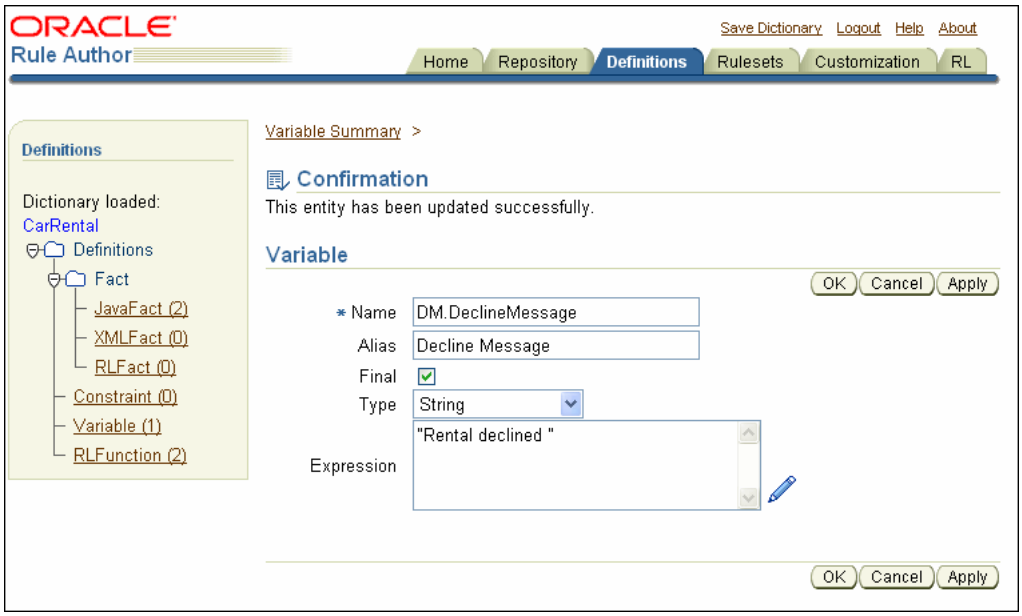
変数を追加する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、CarRental ディクショナリをロードします。
2. **「定義」**タブをクリックします。
3. ナビゲーション・ツリーで、**「変数」**ノードをクリックします。「変数サマリー」ページに、テキスト（アイテムが見つかりません）を含む表が表示されます。これは、変数が何も定義されていないことを示します。
4. **「作成」**をクリックします。「変数」ページが表示されます。
5. **「名前」**フィールドに `DeclineMessage` と入力します。
6. **「エイリアス」**フィールドに `Decline Message` と入力します。
7. **「ファイナル」**チェック・ボックスを選択します（デフォルトでは、このボックスが選択されます）。
8. **「型」**ボックスで、`String` を選択します。
9. **「式」**ボックスに `"Rental declined "` と入力します。

   式の作成を支援するウィザードを使用するには、編集アイコンをクリックして、ウィザードを表示します。
10. **「適用」**をクリックします。Rule Author は確認メッセージを表示します（図 3-1 を参照）。

注意： Rule Author で変数が作成されると、**「名前」**フィールドに入力した名前に `DM.` が追加されます。`DM` は、Data Model（データ・モデル）を表します。

図 3-1 Rule Author の変数を定義するページ



Rule Author の変数を作成する際の注意：

- **「ファイナル」**チェック・ボックスの選択を解除することは、Assign アクションなどで、変数が変更可能であることを示します。
- ルールのテストに使用できるのは、ファイナル変数として指定した変数のみです（非ファイナル変数は使用できません）。

関連項目： ルールのテスト例は、2-19 ページの「UnderAge ルールでのパターンに対するテストの定義」を参照してください。

3.2 制約の使用

カスタマイズ可能ルールで、フィールドへの許容値を特定の値セットに制約する（たとえば、値の範囲を指定する）場合は、Rule Author の制約定義を使用できます。

Rule Author では、表 3-1 に示すように、3 つのタイプの制約定義がサポートされています。

表 3-1 Rule Author の制約タイプ

制約タイプ	説明
範囲	数値の範囲を指定します。
列挙	指定可能な値のリストを指定します。
正規表現	文字列値が準拠する正規表現を指定します。この制約での正規表現の構文は、Java の正規表現の定義に従います。

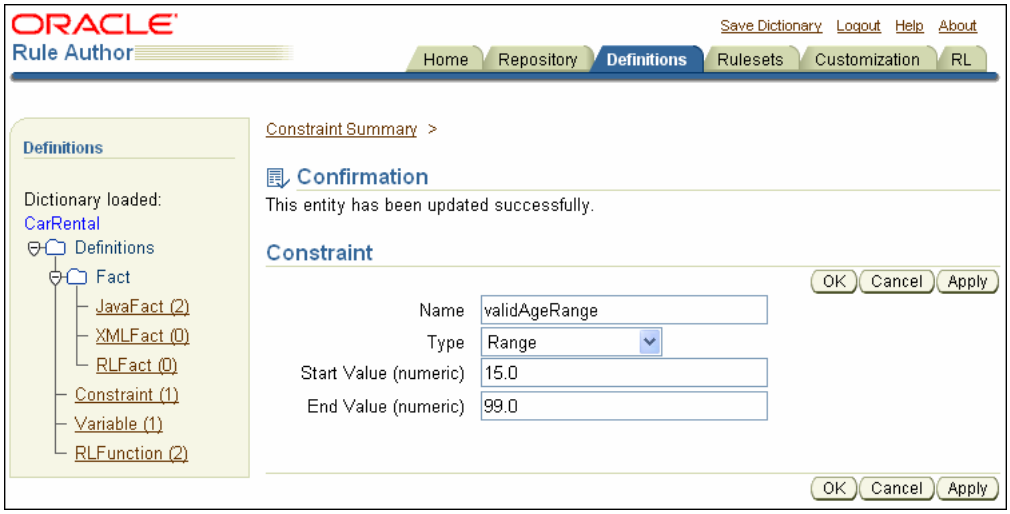
注意： 正規表現の制約では、文字列を引用符で囲む必要があります。

この項の例では、制約を定義して CarRental ディクショナリの UnderAge ルールに追加します。

範囲制約を定義する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、CarRental ディクショナリをロードします。
2. **「定義」**タブをクリックします。
3. ナビゲーション・ツリーで、**「制約」**ノードをクリックします。「制約サマリー」ページに、制約が定義されていないことを示す表が表示されます。
4. **「作成」**をクリックします。「制約」ページが表示されます。
5. **「名前」**フィールドに `validAgeRange` と入力します。
6. **「型」**リストから `Range` を選択します。これにより、「制約」ページの表示が更新されて、**「開始値 (数値)」**と**「終了値 (数値)」**の 2 つの新規フィールドが表示されます。
7. **「開始値 (数値)」**フィールドに `15` を入力します。
8. **「終了値 (数値)」**フィールドに `99` を入力します。
9. **「適用」**をクリックします。Rule Author によって確認メッセージが表示されます（図 3-2 を参照）。

図 3-2 Rule Author の制約を定義するページ



次に、validAgeRange を UnderAge ルールに追加することによってこの制約を使用します。

UnderAge ルールでこの制約を使用する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、CarRental ディクショナリをロードします。
2. **「ルールセット」**タブを選択します。
3. ツリーで、UnderAge ルールを表示するノードを選択します。
4. **「if」**ボックスで編集アイコンを選択します。「パターン定義」ページが表示されます。
5. 「パターン定義」ページの「制約」フィールドで、validAgeRange（「値」列の 2 番目のボックス）を選択します。
6. **「OK」**をクリックします。「パターン定義」ページが閉じます。
7. 「ルール」ページで**「OK」**をクリックします。
8. ディクショナリを保存します。

「カスタマイズ」タブを使用して、Rule Author では指定した範囲の値のみを入力でき、無効な入力は拒否されることを確認します。

注意： ルールセットで使用する制約を変更した場合、そのルールセットは、すべての制約に準拠しなくなる可能性があっても引き続き保存できます。

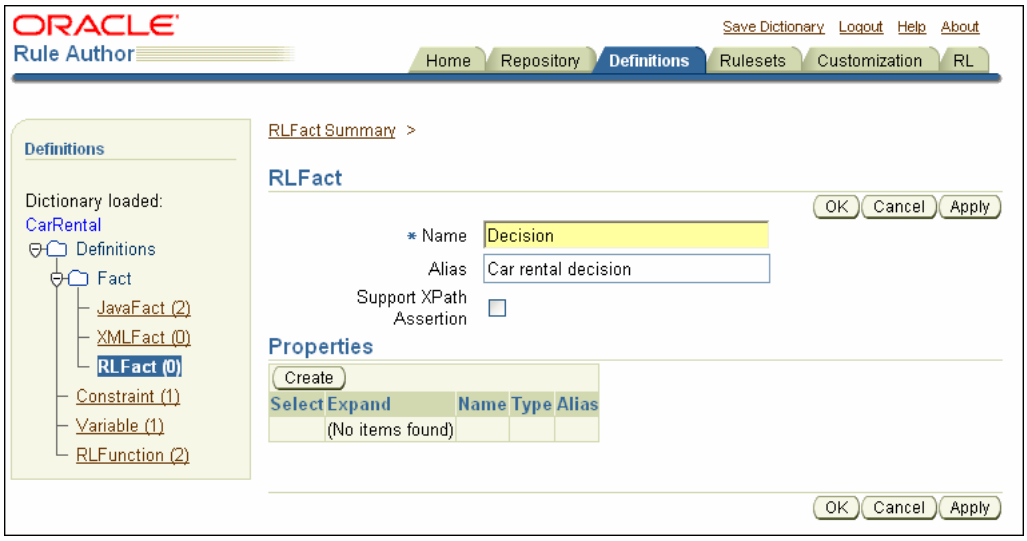
関連項目： 「許可された値」フィールドで制約を使用する方法については、2-19 ページの「UnderAge ルールでのパターンに対するテストの定義」を参照してください。

3.3 RL ファクトの使用

この例では、CarRental ルールを拡張する Decision という RL ファクトを作成します。RL ファクトには、driverName、type および message という 3 つの文字列型のメンバーがあります。RL ファクト Decision を作成する手順は、次のとおりです。

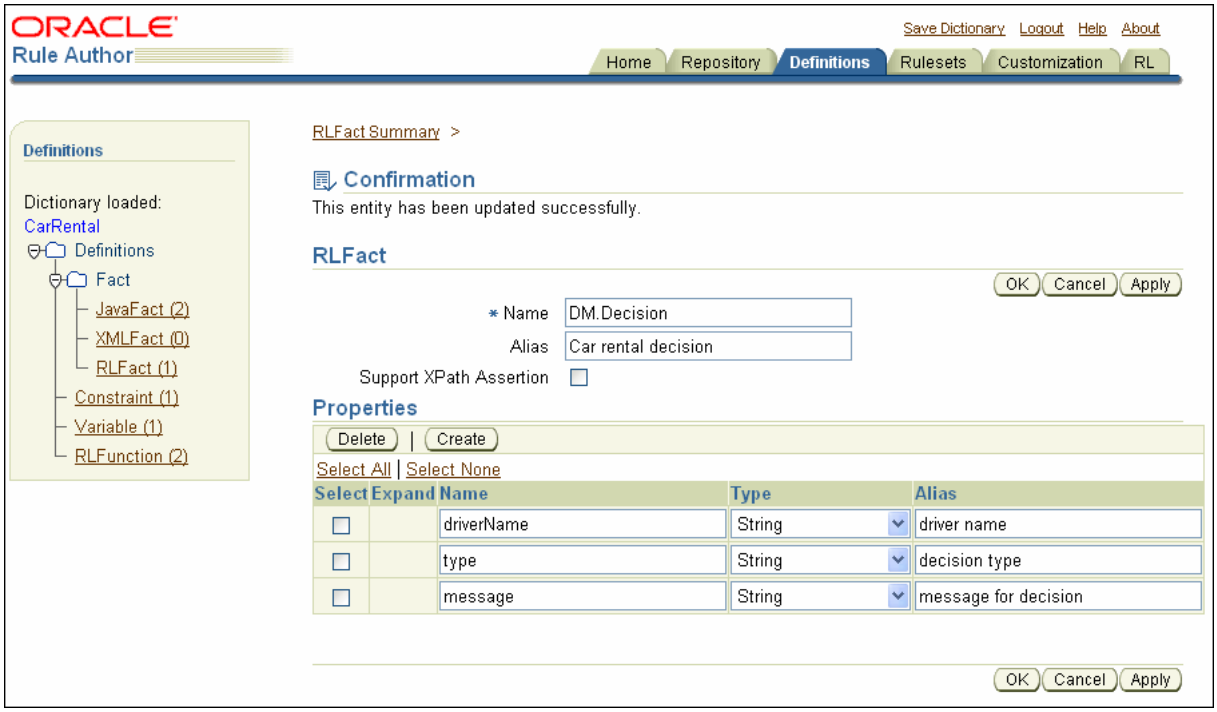
1. **「リポジトリ」**タブをクリックし、CarRental ディクショナリをロードします。
2. **「定義」**タブをクリックします。「ファクト」の下のナビゲーション・ツリーで、**「RL ファクト」**ノードをクリックします。RL ファクトが定義されていない「RL ファクト・サマリー」ページが表示されます。
3. **「作成」**をクリックします。その結果、RL ファクト・ページが表示されています。
4. **「名前」**フィールドに Decision と入力します。
5. 「エイリアス」フィールドに Car rental decision と入力します。図 3-3 を参照してください。

図 3-3 Rule Author の「定義」タブの RL ファクト・ページ



6. 「プロパティ」表で**「作成」**をクリックします。「プロパティ」表に新しい行が表示されます。
7. **「名前」**フィールドに driverName と入力します。
8. **「型」**フィールドのボックスから String を選択します。
9. **「エイリアス」**フィールドに driver name と入力します。
10. **「作成」**をクリックします。「プロパティ」表に別の新規行が追加されます。
11. **「名前」**フィールドに type と入力します。
12. **「型」**フィールドのボックスから String を選択します。
13. **「エイリアス」**フィールドに decision type と入力します。
14. **「作成」**をクリックします。「プロパティ」表に別の新規行が追加されます。
15. **「名前」**フィールドに message と入力します。
16. **「型」**フィールドのボックスから String を選択します。
17. **「エイリアス」**フィールドに message for decision と入力します。
18. **「適用」**をクリックします。Rule Author によって確認メッセージが表示されます（図 3-4 を参照）。

図 3-4 Rule Author の「定義」タブの「RL ファクト・プロパティ」



19. ナビゲーション・ツリーで、**「RL ファクト」**をクリックします。「RL ファクト・サマリー」ページに新規の RL ファクト `DM.Decision` が表示されます。

注意： Rule Author で RL ファクトが作成されると、**「名前」**フィールドに入力した名前に対し `DM.` が追加されます。`DM` は Data Model（データ・モデル）を表します。

関連項目： 図 3-4 で示した**「拡張」**フィールドについては、3-10 ページの「Rule Author のリストに関する参照可能性およびオブジェクト・チェーンの指定」を参照してください。

3.4 関数の使用

Oracle Business Rules では、組込み関数またはユーザー定義関数をルールの条件とアクションに使用できます。この項では、Rule Author を使用して showDecision という関数を定義します。この関数を使用すると、Java 版 How-To に関する結果を出力できます。

注意 1: この項の例では、3.3 項で定義した CarRental ディクショナリと RL ファクトを使用します。

注意 2: Rules SDK から生成された RL Language の場合、たとえば Rule Author を使用してルールを作成すると、グローバル変数が RL Language 関数で直接参照されないことがあります。詳細は、D.2 項「RL 関数で使用できないグローバル変数」を参照してください。

---

showDecision 関数を定義する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、CarRental ディクショナリをロードします。
2. **「定義」**タブを選択します。
3. ナビゲーション・ツリーで**「RL 関数」**を選択します。「RL 関数サマリー」ページが表示されます。
4. **「作成」**をクリックします。
5. **「名前」**フィールドに showDecision と入力します。
6. **「エイリアス」**フィールドに Show Decision と入力します。

注意 : Rule Author で関数を定義している場合は、関数本体で実際の関数名（別名でない）を使用する必要があるときでも、**「エイリアス」**フィールドには有効な別名を指定する必要があります。

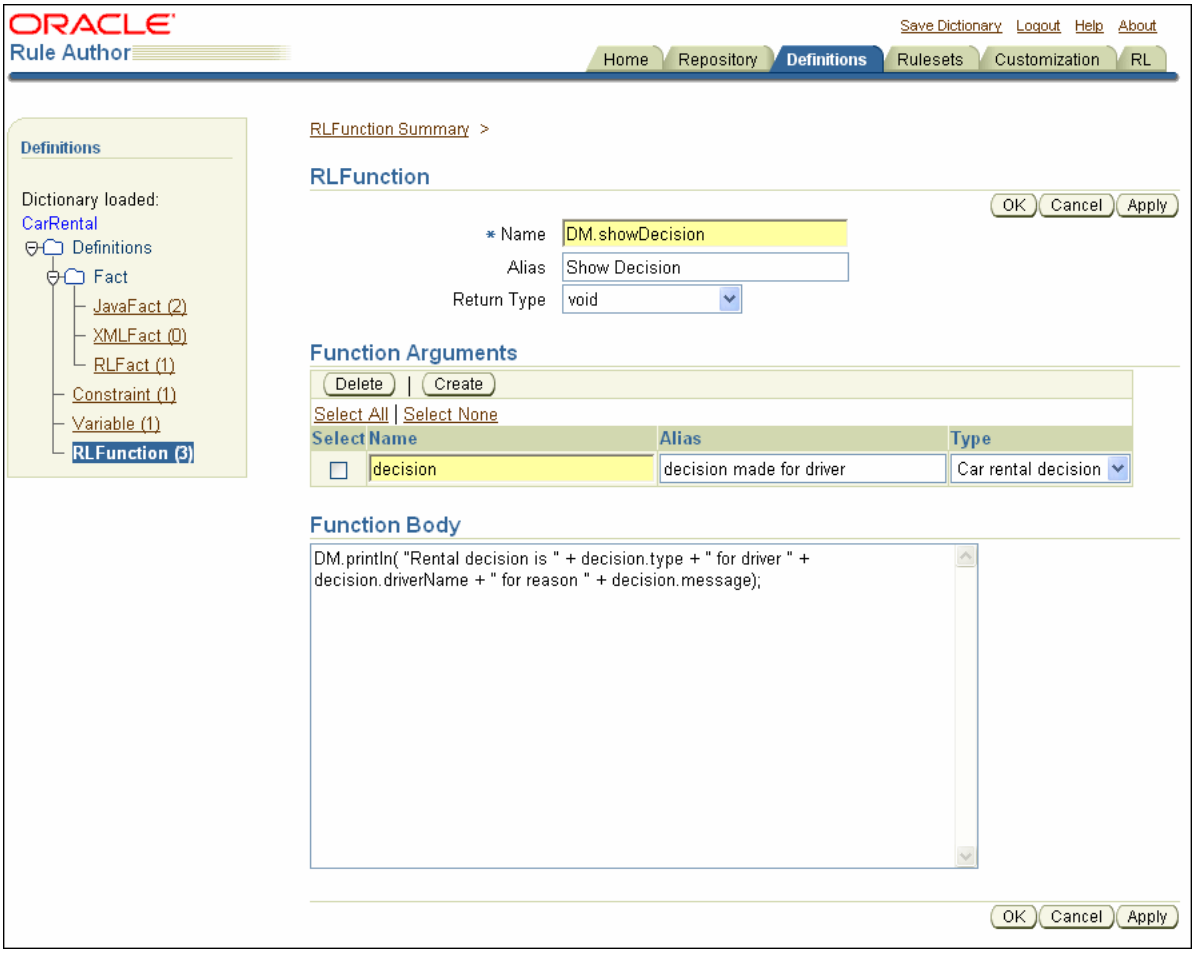
---

7. **「戻り型」**フィールドのボックスで void（デフォルト値）を選択します。
8. **「関数の引数」**表で**「作成」**をクリックします。
9. **「名前」**フィールドに decision と入力します。
10. **「エイリアス」**フィールドに Decision made for driver と入力します。
11. **「型」**フィールドのボックスで、RL ファクト Decision の別名である Car rental decision を選択します。
12. **「関数本体」**ボックスに次の内容を入力します。

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

13. **「適用」**をクリックします。確認メッセージが表示されます。
14. ナビゲーション・ペインで**「RL 関数」**ノードをクリックします。サマリー表に、RL 関数 DM.showDecision が表示されます。
15. 「編集」をクリックして関数を表示します（図 3-5 を参照）。

図 3-5 Rule Author の RL 関数ページ



3.3 項で説明したように新規 RL ファクト `Decision` を作成し、新規 RL 関数 `DM.showDecision` を作成した後は、UnderAge ルールを更新して新規 Decision ファクトを作成するアクションを提供できます。Decision ファクトを `showDecision` 関数で使用するには、Decision ファクトをチェックし、`showDecision` 関数を使用して結果を表示するためのアクションを提供する新規ルールを作成する必要があります。

3.5 ルールの使用

ルールにアクションを追加する場合、アクションにはパターン一致が関連付けられます。ルールの「if」部が一致すると、ルール・エンジンによって「then」部がアクティブになり、ルールに関連付けられたアクションの実行が準備されます。表 3-2 は、ルールを作成するときに使用できるアクションのタイプを示しています。

表 3-2 アクション・タイプ

アクション・タイプ	説明
Assert	パターンで使用されるファクトをアサートします。パターンで一致したファクトが変更された場合、そのファクトをもう一度アサートして、ルール・エンジンに対してファクトが変更されたことを通知する必要があります。
Assert New	新規ファクト・タイプ・インスタンスを作成して、そのインスタンスをルール・エンジンに対してアサートします。
Assign	変数またはファクト・プロパティに値を割り当てます。新しい値がファクト・プロパティに割り当てられた場合は、その新しい値によってルールを再評価するために、ファクトをもう一度アサートする必要があります。
Call	いくつかのアクションを実行する関数をコールできるようにします。
Retract	次のような様々な理由で、ファクトの取消しが必要になることがあります。ファクトの使用が終了し、Rules Engine から削除する場合。ルールに関連付けられているアクションによって状態が変化したため、Rules Engine の現在の状態が示されるように、ファクトを取り消す必要がある場合。
RL	直接実行される自由形式の RL テキストを作成します。この RL の構文は SDK によって検証されないため、RL アクションを作成すると、ルールセットから無効な RL コードが生成されることがあります。

3.6 データ・モデル内の Java オブジェクトの表示

データ・モデル内のオブジェクト（インポートするクラスまたはパッケージを含む）を表示する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、適切なディクショナリをロードします。たとえば、CarRental ディクショナリをロードします。
2. **「定義」**タブをクリックして「定義」ページを表示します。
3. ナビゲーション・ツリーで「ファクト」フォルダを開き、**「Java ファクト」**ノードをクリックして「Java ファクト・サマリー」ページを表示します。

   レンタカー・サンプルでは、インポートされた `carrental.Driver` クラスを含む表が表示されます。
4. 編集アイコンをクリックして、Java ファクトのプロパティとメソッドの表を表示します。

表 3-3 および表 3-4 は、Java ファクトのプロパティおよびメソッドの表のフィールドを説明しています。

表 3-3「Java ファクト・サマリー」のフィールド

フィールド	説明
名前	Java オブジェクトの名前。
エイリアス	Java オブジェクトに指定された別名。Rule Author のリストに表示されます。

表 3-3「Java ファクト・サマリー」のフィールド（続き）

フィールド	説明
参照可能	このボックスは、Java オブジェクトを Rule Author のリストに表示するかどうかを指定します。
XPath アサーションのサポート	このボックスは、XPath 式でクラスを使用して XML データをルール・セッションにアサートできることを示します。

表 3-4「Java ファクト」の「プロパティ」と「メソッド」のフィールド

フィールド	説明
参照可能	プロパティまたはメソッドを Rule Author のリストに表示するかどうかを指定します。
拡張	スーパークラスのあるプロパティまたはメソッドのスーパークラスを Rule Author のリストに表示するかどうかを指定します。
メンバー変数名	プロパティ用。プロパティ名を指定します。
タイプ	プロパティの型を指定します。
エイリアス	変更可能なテキスト・フィールド。プロパティまたはオブジェクトに対するビジネス用語を指定します。指定した名前は、オブジェクトが Rule Author のリストに表示されるときに使用されます。
メソッド名	メソッド用。
引数の型	メソッド用。
戻り型	メソッド用。

注意： Java クラスのインポートでは、そのスーパークラスと、フィールドおよびメソッドを介して関連付けられたクラスが、データ・モデルにインポートされます。「Java ファクト・サマリー」ページの表には、インポートしたクラスのスーパークラスおよび関連付けられたクラスが表示されます。

3.6.1 Rule Author のリストに関する参照可能性およびオブジェクト・チェーンの指定

プロパティ、クラスまたはメソッドを Rule Author の選択ボックスで参照可能にするかどうかを指定できます（プロパティ、クラスまたはメソッドが含まれる選択ボックスは、**「ルールセット」**タブでルールを作成するときに表示されます）。

注意： Java How-To の場合、オブジェクト・チェーンの変更は不要です。

Java オブジェクトの参照可能性を削除する手順は、次のとおりです。

1. 「Java ファクト」ページの上部にある**「参照可能」**ボックスを使用して、オブジェクトを参照可能にするかどうかを指定します（デフォルトでは、オブジェクトは参照可能です）。
2. **「参照可能」**ボックスの選択を解除すると、Rule Author の選択ボックスからオブジェクトが削除されます。
3. 「Java ファクト」ページで**「OK」**をクリックします。

Java プロパティまたは Java メソッドの参照可能性を削除する手順は、次のとおりです。

1. 「Java ファクト」ページの「プロパティ」領域または「メソッド」領域で、プロパティまたはメソッドの選択を解除して、Rule Author の選択ボックスからそのプロパティまたはメソッドを削除します。
2. 「Java ファクト」ページで**「OK」**をクリックします。

Rule Author の選択ボックスに、スーパークラス・チェーン内の指定したメソッドまたはプロパティの 1 レベル上のメソッドまたはプロパティを表示するには、「Java ファクト」ページで、そのメソッドまたはプロパティの**「拡張」**ボックスを選択します。この「拡張」ボックスは、「プロパティ」領域または「メソッド」領域の**「拡張」**フィールドに表示されます。**「拡張」**ボックスは、スーパークラスがあるメソッドまたはプロパティに対してのみ表示されます（Rule Author では、プリミティブ型には**「拡張」**ボックスは表示されません）。

3.7 Oracle Business Rules RL Language テキストの生成

Rule Author では、**「RL」**タブを使用して、RL Language テキストを表示できます。このテキストは、データ・モデルおよびディクショナリ・データに関連付けられたルールセットを示します。

3.7.1 RL Language テキストの生成、表示およびチェック

RL Language テキストを生成、表示、およびチェックする手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックし、ディクショナリをロードします。たとえば、CarRental ディクショナリをロードします。
2. **「RL」**タブを選択します。
3. ナビゲーション・ツリーで、目的のルールセットを選択します。
4. **「RL の生成」**をクリックします。指定したルールセットの RL Language テキストが表示されます。
5. **「RL 構文のチェック」**をクリックして、RL Language テキストを検証します。

注意： Rule Author の **RL 構文チェック**機能を使用する場合、データ・モデルに Java クラスが含まれている場合は、その Java クラスを OC4J クラスパスに含めて検証を実行する必要があります。また、データ・モデルに XML スキーマが含まれている場合は、生成された JAXB クラス・ファイルを OC4J クラスパスに含めて検証を実行する必要があります。

したがって、検証を実行するには、OC4J クラスパスに Java オブジェクト用あるいは XML スキーマから生成された Java クラス用の jar ファイル、またはクラスが含まれていることを確認する必要があります。

JAR ファイルの Java クラスの場合は、JAR ファイルを次のディレクトリにコピーし、OC4J を再起動できます。

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

Java クラスを共有ライブラリとして組み込むこともできます。その結果、Rule Author では他のアプリケーションとクラスを共有できるようになります。

関連項目： Enterprise Manager を使用して Java クラスを共有ライブラリとして追加する方法については、3-16 ページの「テスト・ルールセットの使用」を参照してください。

3.8 Rule Author のディクショナリ・プロパティの構成

Rule Author では、ディクショナリ・プロパティを使用して、式で使用するデフォルトの式タイプを指定して、ロギングのオプションを指定できます。

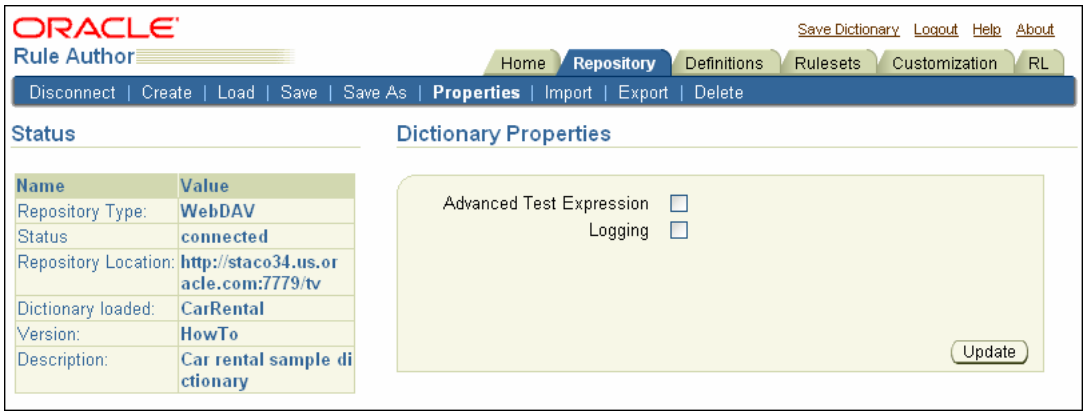
内容は次のとおりです。

- 「アドバンスト・テスト式」オプションの使用
- 「ロギング」オプションの使用

ディクショナリ・プロパティを構成するには、次の手順を実行します。

1. リポジトリに接続していること、およびディクショナリがロードされていることを確認します。
2. **「リポジトリ」**タブをクリックします。
3. **「プロパティ」**第 2 タブをクリックします（図 3-6 を参照）。

図 3-6 Rule Author の「ディクショナリ・プロパティ」ページ



注意： Rule Author は、ユーザーごとにディクショナリ・プロパティを格納します。設定を変更し、ログアウトして、別のユーザーとしてログインすると、ディクショナリ・プロパティには、新しくログインしたユーザーのプロパティが反映されます。

Rule Author で認証を使用しない場合は（ユーザーが `web.xml` でセキュリティを無効にするように構成している場合に可能）、ユーザー・レベルのディクショナリ・プロパティの保存も無効になります。

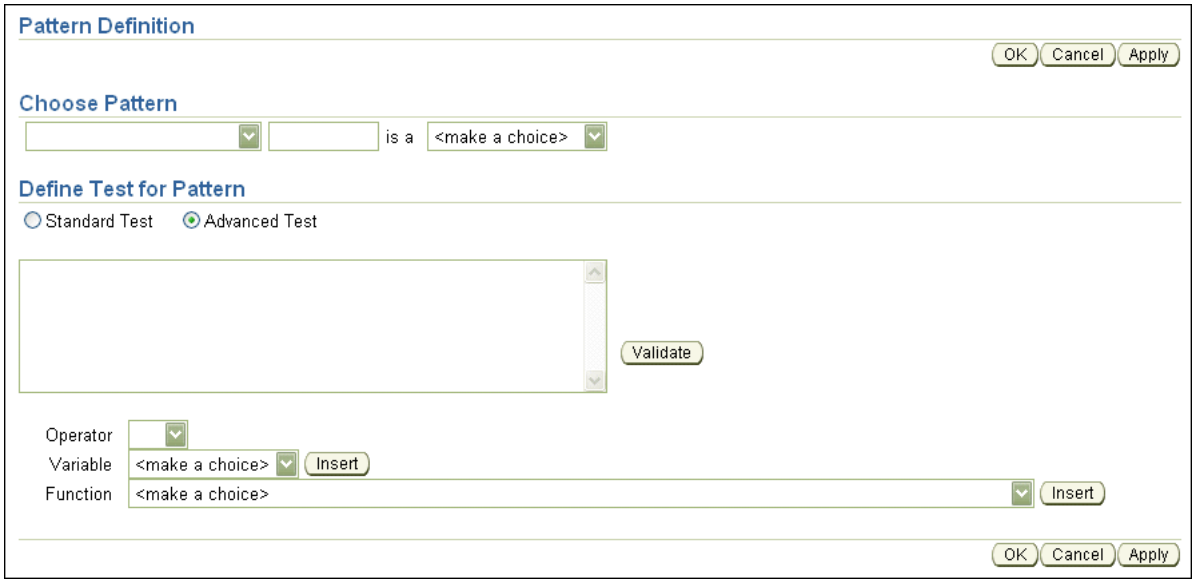
3.8.1「アドバンスト・テスト式」オプションの使用

「アドバンスト・テスト式」チェック・ボックスによって、テスト式に対する Rule Author の式モードが拡張に変更されます。このテスト式は、ルールのパターンを編集するときに表示されます（図 3-7 を参照）。ルール条件のテストには、数学的演算や論理積が伴う場合があります。Rule Author には、このような複雑な式の定義をサポートする拡張式モードがあります。

パターンを初めて作成するときのテスト式モードは、「プロパティ」ページで設定した**「アドバンスト・テスト式」**プロパティによって決まります。「アドバンスト・テスト式」チェック・ボックスを選択すると、すべての新規パターンにアドバンスト・テスト式モードが適用されます。この設定は、ディクショナリの保存時に保持されます。この場合は、ディクショナリのロード時に、すべてのパターンがアドバンスト・テスト式モードで作成されます。作成後のパターンは、テストの有無に関係なく、テスト式モードに永続的に関連付けられます。したがって、パターンに関連するテスト式モードは変更できません。

単一ルール内で、基本式モードと拡張式モードで作成した両方のパターンを使用できます。

図 3-7「アドバンスト・テスト式」ページの「パターン定義」



3.8.2「ロギング」オプションの使用

「ロギング」ボックスによって、ロギングのオプションが指定されます。このオプションは、Rule Author の問題をレポートする必要がある場合に便利です。ロギングを指定するには、**「ロギング」**チェック・ボックスを選択した後、次のログ・ファイル・プロパティを選択します。

- **ログ・レベル**：**「エラー」**は最低ログ・レベルで、最少量の出力を生成します。**「ステータス」**は標準ログ・レベルです。**「トレース」**は最高ログ・レベルで、最大量の出力を生成します。
- **「ログ・ディレクトリ」**ボックスを使用して、ログを保存するディレクトリを指定します。**「ログ・ディレクトリ」**ボックスでのログ・ディレクトリの指定はオプションです（ディレクトリが指定されていない場合、ログはコンソールに表示されます）。
- **「ログ・ファイル名」**を指定すると、*<log_file_name>.<last_8_session_id>* という名前のファイルにログが保存されます。たとえば、ログ・ファイル名に `RALog` を指定し、セッション ID の最後の 8 桁が `11223344` の場合は、`RALog.11223344` というファイルが作成されます。ログ・ファイルが指定されていない場合、ログは、`RuleAuthor.`*<last_8_session_id>* という名前のファイルに保存されます。

ロギング・オプションの指定が終了した後、**「更新」**をクリックします。

3.9 Rule Author のディクショナリの削除

この項では、ディクショナリのバージョンまたはディクショナリ全体を削除する方法について説明します。

個々のディクショナリ・バージョンを削除する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「削除」**第 2 タブをクリックします。

   特定のディクショナリ・バージョンを削除する場合は、「ディクショナリ・バージョンの選択」セクションでディクショナリとバージョンを選択し、**「バージョンの削除」**をクリックします。

   ディクショナリ全体（およびそのすべてのバージョン）を削除する場合は、「ディクショナリ全体の選択」セクションでディクショナリを選択し、**「削除」**をクリックします。

3.10 ディクショナリのインポートとエクスポート

Rule Author には、ディクショナリの特定バージョンまたはディクショナリ全体をインポートできます。手順は次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「インポート」**第 2 タブをクリックします。

   ディクショナリがローカルにある場合は、最初の箇条書きのセクションを使用してその場所を指定します。ディクショナリのパスを手動で入力するか、または「参照」ボタンをクリックしてディクショナリを選択できます。

   ディクショナリが（Rule Author が実行されていない）別のマシンにある場合は、そのサーバー上のディクショナリのフルパスを指定する必要があります。

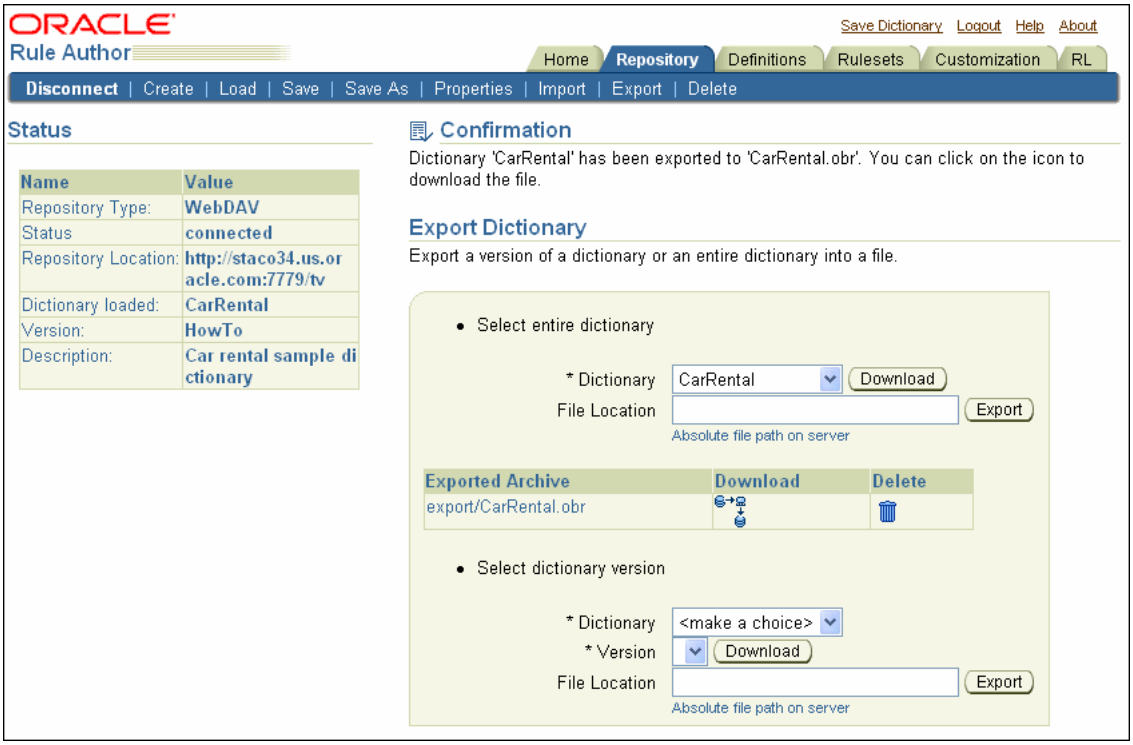
3. **「インポート」**をクリックします。

ディクショナリ全体をエクスポートする手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「エクスポート」**第 2 タブをクリックします。
3. 「ディクショナリ全体の選択」セクションで、次の操作を実行します。
   a. 「ディクショナリ」フィールドで、エクスポートするディクショナリを選択します。
   b. 「ファイルの場所」フィールドで、ディクショナリをエクスポートする場所とファイル名（サーバー上の絶対ファイル・パス）を指定します。
4. **「エクスポート」**をクリックします。

   ディクショナリを選択して**「ダウンロード」**をクリックすることもできます。これによって、エクスポート済アーカイブへのリンクが「ディクショナリのエクスポート」ページに作成されます。次に、リンクをクリックすると、ブラウザを使用して、選択した場所にアーカイブをダウンロードできます（図 3-8 を参照）。

図 3-8 Rule Author の「ディクショナリのエクスポート」ページ



特定のディクショナリ・バージョンをエクスポートする手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「エクスポート」**第 2 タブをクリックします。
3. 「ディクショナリ・バージョンの選択」セクションで、次の操作を実行します。
   a. 「ディクショナリ」フィールドで、エクスポートするディクショナリを選択します。
   b. 「バージョン」フィールドで、エクスポートするディクショナリ・バージョンを選択します。
   c. 「ファイルの場所」フィールドで、ディクショナリをエクスポートする場所とファイル名（サーバー上の絶対ファイル・パス）を指定します。
4. **「エクスポート」**をクリックします。

   ディクショナリとバージョンを選択して、**「ダウンロード」**をクリックすることもできます。これにより、「ディクショナリのエクスポート」ページのエクスポート済アーカイブへのリンクが作成されます。次に、ブラウザを使用して、選択した場所にアーカイブをダウンロードできます（図 3-8 を参照）。

注意： ディクショナリのエクスポートは追加されていく操作です。したがって、ディクショナリのバージョンをエクスポートし、その後、同じ jar ファイルに同じディクショナリの別のバージョンをエクスポートすると、エクスポート操作の後、その jar ファイルには同じディクショナリの両方のバージョンが含まれることになります。

たとえば、ディクショナリ dict1 に v1 および v2 という 2 つのバージョンがあり、バージョン（dict1、v1）を `myExport.jar` にエクスポートした場合、その jar ファイルには v1 の内容のみが含まれます。さらに同じファイル `myExport.jar` に対して（dict1、v2）のエクスポートを実行すると、2 番目のエクスポートの完了後、このファイルには dict1 の v1 および v2 の両方のエクスポート済バージョンが含まれます。

3.11 テスト・ルールセットの使用

Rule Author では、テスト・ルールセット機能により、RL 関数を使用してルールセットをテストできます。選択したルールセットとそのルールセットに関連付けられたデータ・モデルがルール・セッションで実行され、ルールを実行するユーザー定義関数がコールされます。

テスト・ルールセット機能を使用する手順は、次のとおりです。

1. テストするルールセットを作成します。データ・モデルに Java クラスまたは XML スキーマが含まれている場合は、その Java クラスまたは生成された JAXB クラスを OC4J クラスパスに含める必要があります。

   Java クラスの場合は、次のディレクトリに JAR ファイルを挿入してから、OC4J を再起動してクラスパスにクラスを追加できます。

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

   Java クラスを共有ライブラリとして組み込むこともできます。その結果、Rule Author では他のアプリケーションとクラスを共有できるようになります。これを実行するには、Enterprise Manager にログインして次の操作を実行します。

   a. 「OC4J: ホーム」ページにナビゲートします。

   b. **「管理」**タブをクリックします。

   c. **「プロパティ」**ノードの下で 「共有ライブラリ」を検索し、「タスクに移動」列のアイコンをクリックします。

   d. **「作成」**をクリックし、ライブラリ名とバージョン番号を入力した後、**「次」**をクリックします。

   e. **「追加」**をクリックして JAR ファイルの場所にナビゲートし、**「続行」**をクリックします。

   f. **「終了」**をクリックして「共有ライブラリ」ページに戻ります。

   g. 作成したライブラリのリンクを検索してクリックします。アーカイブの絶対パスをコピーします。

   h. 「OC4J: ホーム」ページに戻ります。

   i. **「アプリケーション」**タブをクリックします。

   j. Rule Author アプリケーションのリンク（Rule Author アプリケーションを初めてデプロイしたときに定義されています）をクリックします。

   k. **ruleauthor** モジュール・リンクをクリックします。

   l. **「管理」**タブをクリックします。

   m. **「構成プロパティ」**ノードを検索し、「タスクに移動」アイコンをクリックします。

n. 「クラスパス」フィールドに、ステップ g でコピーしたアーカイブの絶対パスを貼り付けて、**「OK」**をクリックします。

o. 結果の確認メッセージが表示された後、**「再起動」**リンクをクリックして Rule Author を再起動します。

2. 次のようにテスト関数を作成します。

a. **「定義」**タブをクリックします。

b. ナビゲーション・ツリーで、**「RL 関数」**ノードをクリックします。

c. 「名前」フィールドと「エイリアス」フィールドに `test` と入力します。

d. 戻り型は `void` のままにします。

e. 「関数本体」フィールドに次の内容を入力します。

```
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();
```

この関数は、複数のファクトをアサートし、`vehicleRent` ルールセットをルールセット・スタックに追加し、`run()` をコールします。

注意： スタックには、実行するすべてのルールセットを追加する必要があります。これらのルールセットは、単に選択するのみでは自動的に追加されません。

ロギングを有効にするには、`watchFacts()` 関数または `watchActivations()` 関数をコールできます。

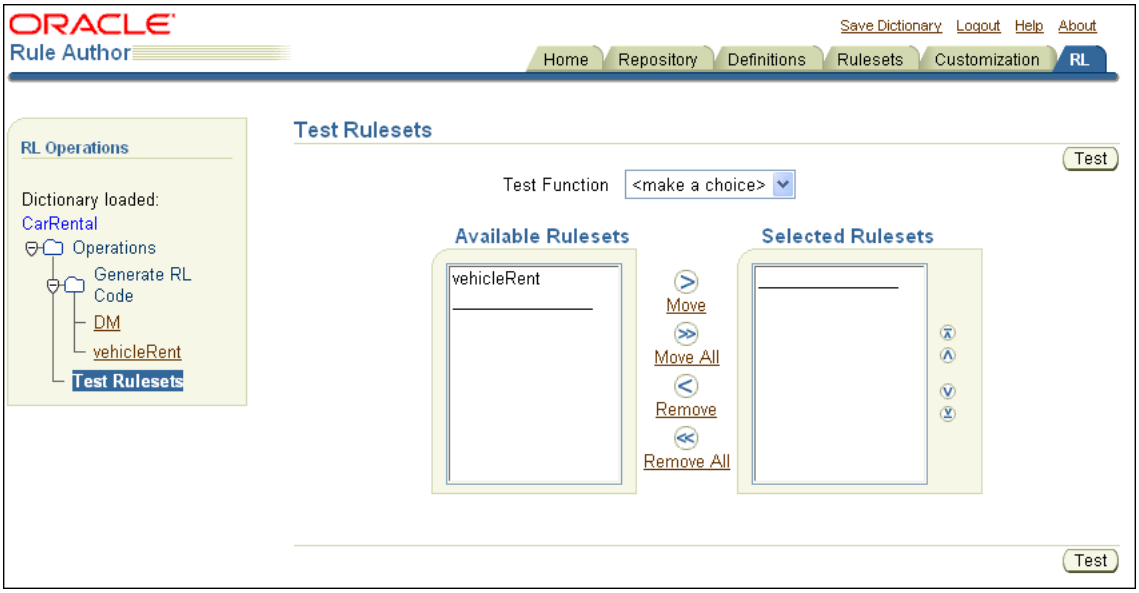
f. **「OK」**をクリックします。これにより、RL 関数 `DM.test` が保存されます。

g. ディクショナリを保存します。

3. **「RL」**タブをクリックします。

4. ナビゲーション・ツリーで、**「テスト・ルールセット」**ノードをクリックします。「テスト・ルールセット」ページが表示されます。

図 3-9 Rule Author の「テスト・ルールセット」ページ

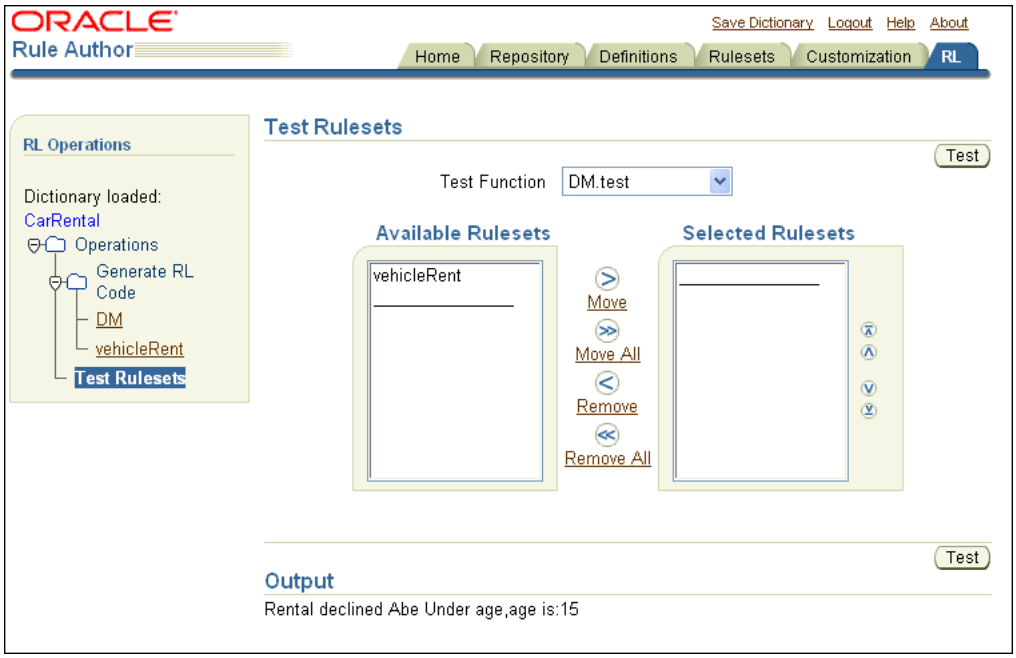


`vehicleRent` ルールセットが使用可能であることを確認できます。このルールセットは、第 2 章で作成したものです。

5. `vehicleRent` ルールセットを選択して「選択したルールセット」列に移動します。
6. 「テスト関数」リストから**「DM.test」**関数を選択します。
7. **「テスト」**をクリックします。

   選択したルールセットに対して RL Language コードが生成され、ルール・セッションに挿入されます。次に、選択したテスト関数がコールされます。関数の出力が画面に表示されます（図 3-10 を参照）。

図 3-10「出力」が表示されている Rule Author の「テスト・ルールセット」ページ



3.12 ルールの起動と Rules Engine 結果の取得

ルール対応プログラムでは、通常、次の手順でルールを起動します。

1. ファクトとしてアサートするオブジェクトをルール・エンジンに渡します。
2. ルールを実行します。
3. 起動したルールから生成された結果を取得します。

この項では、RL 関数を使用してルールの起動をカプセル化するためのベスト・プラクティスについて説明します。ルールを起動する 3 つの方法について説明します。これらの方法の主な相違点は、結果を取得する方法の詳細にあります。この項では、`getSubscribers` というサンプル RL 関数を使用します。このルーチンを使用すると、`getSubscribers` メソッドをコールするルール対応プログラムという観点において、ルール起動の各方法が同じであるように見えます。

内容は次のとおりです。

- 結果の例の概要
- グローバル変数を使用した結果の取得
- コンテナ・オブジェクトを使用した結果の取得
- 判断オブジェクトを使用した結果の取得

3.12.1 結果の例の概要

この項で使用する例は、高速道路の事故通知システムを示しています。それぞれの例は、ルール・エンジンの評価結果にアクセスするための異なる方法を示しています。これらの例では、2 つの Java クラス `traffic.TrafficIncident` および `traffic.IncidentSubscription` を使用します。

注意： サンプルの `traffic.*` クラスは、Oracle Business Rules に同梱されていません。

`TrafficIncident` クラスは、交通状況に影響を与える事故の情報を表し、次のプロパティが含まれています。

- どの高速道路か
- 上りか下りか
- 事故の種類
- 事故発生時刻
- 渋滞の見積時間（分単位）

`IncidentSubscription` クラスは、特定の高速道路における事故の通知へのサブスクリプションを記述し、次のプロパティが含まれています。

- サブスクライバ : サブスクライバの名前
- 該当する高速道路
- 上りか下りか

この例では、高速道路で交通状況に影響を与える事故が発生したとき、これらのクラスを使用して `TrafficIncident` オブジェクトをアサートし、ルール評価によって通知の受信者を判断します。

例に示された `sess` は `RuleSession` オブジェクトで、事故通知の多数のサブスクリプションがアサートされます。単純化するために、`TrafficIncident` オブジェクトは永続的でないことを前提にしています。実際には、このオブジェクトはアサート対象イベントを表し、その時点で登録されているサブスクライバにのみ通知が送信されます。

3.12.2 グローバル変数を使用した結果の取得

結果の累計にはグローバル変数を使用することが最良の方法です。この方法を使用すると、3.12.3 項および 3.12.4 項に示す方法よりも単純なルール条件が生成されます。

例 3-1 は、TrafficIncident オブジェクトをアサートし、Map オブジェクトによってグローバル変数を初期化し、Rules Engine を起動して、結果の Map オブジェクトを返す getSubscribers 関数を示しています。

例 3-1 グローバル変数にアクセスする RL 関数を使用した結果の取得

```
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);
      }
   }
}
```

例 3-2 に、getSubscribers 関数を起動し、結果を出力する Java コードを示します。

例 3-2 グローバル変数の使用による結果の例

```
// 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 コンテナ・オブジェクトを使用した結果の取得

コンテナ・オブジェクト方法では、1 つ以上のオブジェクトが作業メモリーにアサートされ、結果のコンテナとして機能します。オブジェクトをアサートする RL Language コードによって、そのオブジェクトの参照が常に使用できるように保持されます。起動されたルールは、結果をコンテナに追加できます。コンテナ・オブジェクトは、Java コレクション・クラスの 1 つか、特定のアプリケーション固有のコンテナ・オブジェクトの場合があります。ルール評価が完了すると、コンテナ・オブジェクトを検査して結果にアクセスできます。

例 3-3 は、`java.util.Map` オブジェクトを使用し、サブスクライバ（`key`）と事故（`value`）を `Map` オブジェクトに追加する `getSubscribers` 関数を示しています。`getSubscribers` 関数は、`Map` オブジェクトおよび `TrafficIncident` オブジェクトをアサートし、Rules Engine を起動して、結果を `Map` オブジェクトに返します。

注意： `Map` オブジェクトは、更新されている場合でも再アサートされません。通常、判断オブジェクトが更新されると、そのオブジェクトを再アサートする必要があります。このユースケースは、このルールに対する例外を表しています。マップ `alerts` は、単に結果のコンテナであり、その内容は判断に関係しません。このため、再アサートしなくても問題ありません。

例 3-3 コンテナ・オブジェクトを使用した結果の取得

```
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);
      }
   }
}
```

例 3-2 に、`getSubscribers` 関数を起動し、結果を出力する Java コードを示します。

3.12.4 判断オブジェクトを使用した結果の取得

判断オブジェクト方法では、1 つ以上のオブジェクトがルール・エンジンの作業メモリーにアサートされ、オブジェクト参照は（getSubscribers 関数の）RL Language に保持されます。ルール評価プロセスによって、これらのオブジェクトの 1 つ以上が更新されます。これらのオブジェクトは、ルール評価による結果の判定後に検査されます。

例 3-4 では、TrafficIncident クラスが、通知する必要のあるサブスクライバの更新済 java.util.Set を保持するように変更されます。TrafficIncident オブジェクトは判断されるために再アサートされます。同じサブスクリプションに対するルール起動の無限ループを回避するには、subscribed メソッドを使用して、一致した事故をテストする必要があります。このメソッドによってループが防止されます。subscribed メソッドは、一致する TrafficIncident オブジェクトにサブスクライバがすでに追加されている場合は true を戻します。これは、ルール・プログラミングの共通の特徴です。ルール・アクションは判断されるファクトを更新します。不要な追加起動を回避するには、その更新の存在の有無をチェックする条件にテストを追加します。

例 3-4 は、TrafficIncident オブジェクトをアサートし、Rules Engine を起動して、結果の Map オブジェクトを作成して返す getSubscribers 関数を使用するコードを示しています。

例 3-4 判断オブジェクトを使用した結果の取得

```
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);
      }
   }
}
```

例 3-2 に、getSubscribers 関数を起動し、結果を出力する Java コードを示します。

4

Rule Author での XML ファクトの使用

この章では、XML 文書を使用して Oracle Business Rules で作業するためのチュートリアルを提供します（ファクトは XML 文書で提供されます）。また、XML を使用するルール対応 Java アプリケーションの作成方法も示します。

ルールの XML 版 How-To では、XML 文書で提供される運転者データを使用して運転者情報を指定し、この章で作成するビジネス・ルールを使用してレンタカー会社のサービス担当が（定義するレンタカー会社のビジネス・ポリシーに従って）運転者の年齢制限により車両の貸出しを断る必要があるかどうかを判断します。

この章の内容は次のとおりです。

- Rule Author での XML 文書およびスキーマの使用の概要
- Rule Author ユーザーの作成と Rule Author の起動
- XML 版レンタカー・サンプルに対するディクショナリの作成および保存
- XML 版レンタカー・サンプルに対するデータ・モデルの定義
- XML 版レンタカー・サンプルに対するビジネス用語の定義
- XML 版レンタカー・サンプルに対するルールの定義
- XML 版レンタカー・サンプルに対するルールのカスタマイズ
- XML ファクトを使用したルール・セッションがある Java アプリケーションの作成
- テスト・プログラムを使用した XML 版レンタカー・サンプルの実行

注意： この章のサンプル・アプリケーションをルールの XML 版 How-To と呼びます。このサンプル・アプリケーションは、Oracle Technology Web サイトの「Oracle Business Rules」領域の「Viewlets and Tutorials」見出しの下で入手できます。

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

4.1 Rule Author での XML 文書およびスキーマの使用の概要

Rule Author を使用して、XML 要素をデータ・モデルにインポートし、条件式に XML 要素を使用してルールを記述できます。たとえば、アプリケーションに関連するデータが格納されている XML 文書があり、その XML 文書にスキーマが関連付けられている場合は、Rule Author を使用して、指定の要素に基づいて XML スキーマからルールを定義できます。

Rule Author で XML 文書を使用して、XML スキーマに着手する際に必要な手順は、次のとおりです。

1. Rule Author は、Java Architecture for XML Binding（JAXB）パッケージ、クラス、および XML スキーマに対するインタフェースを生成するために、提供される JAXB コンパイラを実行することで、XML スキーマから Java クラスを生成します。
2. ディクショナリのデータ・モデルに XML 要素をインポートします。
3. XML 文書からの XML 要素に基づいてビジネス・ポリシーを指定するルールを定義します。XML 文書のルールを記述するプロセスは、Java オブジェクトのルールを記述する場合とほぼ同じです。

Rule Author でルールを作成した後は、XML 文書でルールを使用するアプリケーションを記述できます。XML 文書の処理を完了するには、XML 文書の要素を Rules Engine セッションにアサートします。

注意： Rule Author で提供される実装とは異なる JAXB バインディング・コンパイラを使用するには、その JAXB バインディング・コンパイラを使用して XML スキーマ処理を手動で実行してから、生成された Java パッケージとクラスをデータ・モデルにインポートできます。

JAXB の詳細は、

`http://java.sun.com/webservices/jaxb/` を参照してください。

関連項目： 2-9 ページ「レンタカー・サンプルでのファクトとしての Java オブジェクトの使用」

4.2 Rule Author ユーザーの作成と Rule Author の起動

Rule Author は、ホーム・ページの URL を入力することで起動します。ホーム・ページの URL には、通常、ホスト・コンピュータ名とインストール時にアプリケーション・サーバーに対して割り当てられたポート番号、および Rule Author のホーム・ページのパスが含まれています。

関連項目： 2-2 ページの「Rule Author ユーザーの作成」を参照してください。

4.3 XML 版レンタカー・サンプルに対するディクショナリの作成および保存

Rule Author での作業は、ディクショナリから着手する必要があります。Rule Author では、ルールおよび関連する定義をディクショナリに格納します。ディクショナリを作成または保存するには、ディクショナリを格納するリポジトリに接続する必要があります。出荷時の Rule Author では、WebDAV（Web Distributed Authoring and Versioning）リポジトリとファイル・リポジトリの 2 種類のリポジトリがサポートされています。

ここでは、WebDAV リポジトリを使用して XML 版 How-To のディクショナリを作成し、保存します。

> **注意：** この章に記載されているディクショナリを作成するには、（WebDAV リポジトリまたはファイル・リポジトリを使用して）新しいディクショナリを作成するか、あるいは How-To とともに提供される *$HowToDir*/dict ディレクトリから完全なディクショナリをロードできます。$*HowToDir* には、XML 版 How-To をインストールしたディレクトリが入ります。

内容は次のとおりです。

- Rule Author リポジトリへの接続
- Rule Author ディクショナリの作成
- Rule Author ディクショナリのバージョン付き保存
- Rule Author ディクショナリの保存

4.3.1 Rule Author リポジトリへの接続

Oracle Business Rules では、ディクショナリには、ルールおよびそのルールに関連付けられているデータ・モデルが格納されます。

> **注意：** 使用するリポジトリ（WebDAV リポジトリまたはファイル・リポジトリ）の選択に関係なく、リポジトリに接続するためには、リポジトリが存在している必要があります。詳細は、付録 B「リポジトリを利用した Rule Author と Rules SDK の使用」を参照してください。

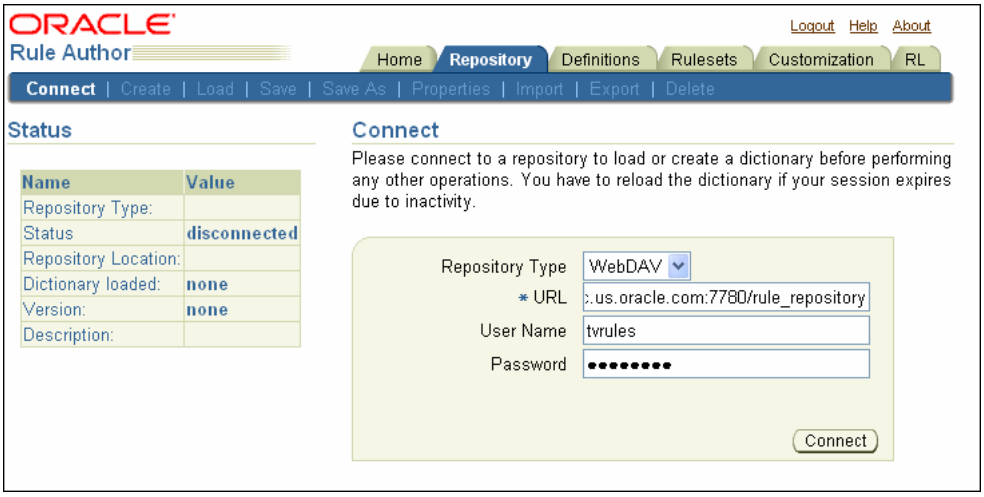
リポジトリに接続する手順は、次のとおりです。

1. **「リポジトリ」**タブをクリックします。
2. **「接続」**第 2 タブをクリックします。
3. **「リポジトリ・タイプ」**フィールドで「WebDAV」リポジトリ・タイプを選択します。
4. URL を WebDAV リポジトリに入力します（図 4-1 を参照）。URL は次の形式で指定する必要があります。

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

   > **注意：** 認証が実行されるように、URL には完全修飾ホスト名を使用する必要があります。

図 4-1 Rule Author の「リポジトリ」タブにある「接続」ページ



WebDAV リポジトリの設定方法の詳細は、B.1 項「WebDAV リポジトリの使用」を参照してください。

5. Rule Author が実行されているサーバーと WebDAV サーバーとの間にプロキシ・サーバーがある場合は、(プロキシ・サーバーでの必要に応じて)**「プロキシ・ユーザー名」**フィールドと**「プロキシ・パスワード」**フィールドに値を指定します。
6. **「接続」**をクリックします。接続すると、Rule Author によって確認メッセージが表示されます。

注意： ファイル・リポジトリの場合、所定の時間にリポジトリを編集できるのは、そのリポジトリに格納されているディクショナリ数に関係なく常に 1 人のユーザーのみです。WebDAV リポジトリの場合は、1 人のユーザーが複数のディクショナリを同時に編集できます。

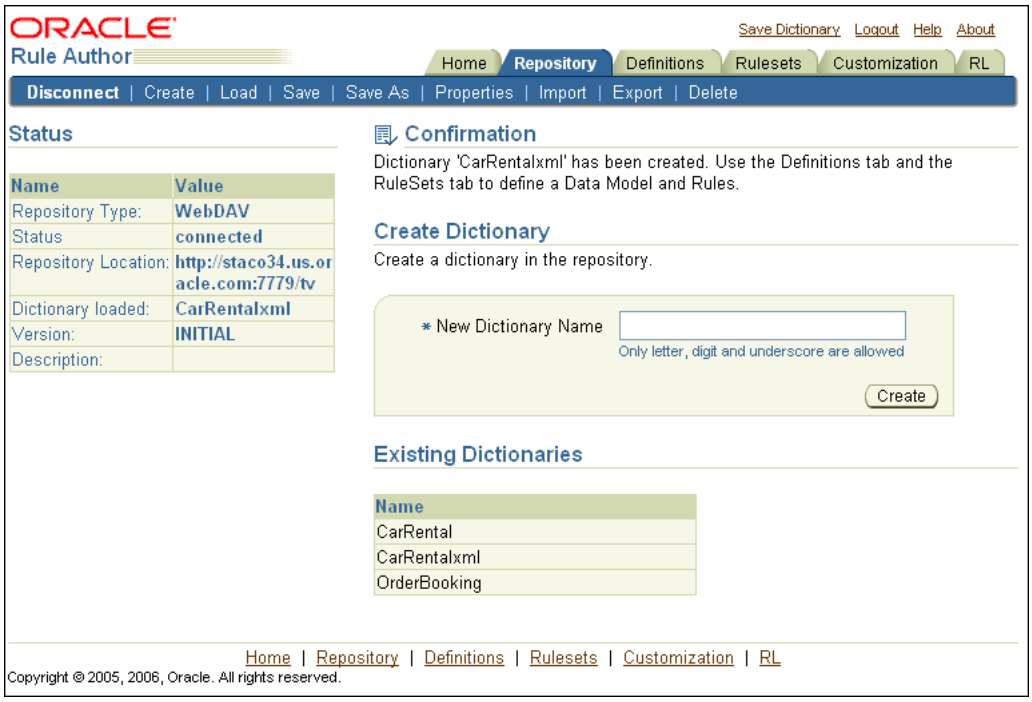
4.3.2 Rule Author ディクショナリの作成

ディクショナリは最上位コンテナであり、Rule Author で作業する起点となります。ディクショナリは、通常、アプリケーションのルール部分に対応しています。

ディクショナリを作成する手順は、次のとおりです。

1. **「リポジトリ」**タブからリポジトリに接続します。
2. **「作成」**第 2 タブをクリックします。
3. **「新規ディクショナリ名」**フィールドにディクショナリ名を入力します。この例では、`CarRentalxml` と入力します。
4. **「作成」**をクリックします。**「作成」**をクリックすると、ステータス・メッセージが表示されます（図 4-2 を参照）。

図 4-2 Rule Author の「ディクショナリを作成」（XML）

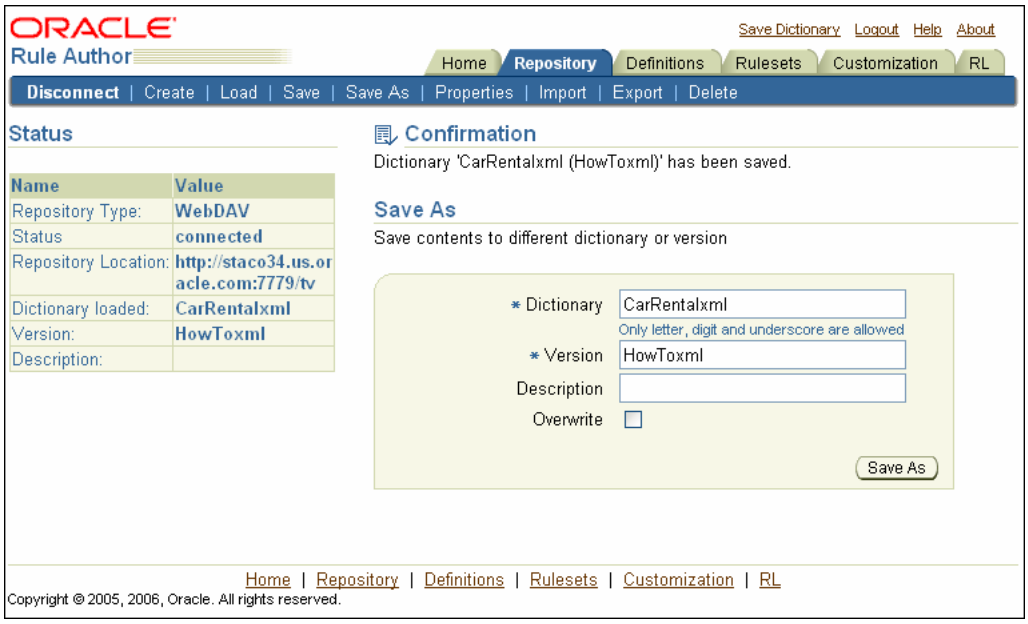


4.3.3 Rule Author ディクショナリのバージョン付き保存

別のディクショナリ名で保存する場合や現行のディクショナリにバージョンを指定する場合は、次のように「別名保存」領域を使用します。

1. **「リポジトリ」**タブをクリックします。
2. **「別名保存」**第 2 タブをクリックします。
3. **「ディクショナリ」**フィールドにディクショナリ名を入力します（`CarRentalxml` など）。
4. ディクショナリに関連付けられているバージョンを指定するには、**「バージョン」**フィールドにテキストを入力します。たとえば、`HowToxml` と入力します。
5. **「別名保存」**をクリックします。**「別名保存」**をクリックすると、Rule Author によって確認メッセージが表示されます（図 4-3 を参照）。

図 4-3 Rule Author の「ディクショナリの保存」（XML）



4.3.4 Rule Author ディクショナリの保存

データの消失を防止するために、ディクショナリを定期的に保存してください。ディクショナリを保存するには、次のいずれかを実行します。

- **「リポジトリ」**タブをクリックし、次に**「保存」**第 2 タブをクリックします。
- ページ上部の**「ディクショナリの保存」**リンクをクリックします。

前述のいずれかの操作を実行した後、「ディクショナリの保存」ページの**「保存」**をクリックします。**「保存」**をクリックすると、ステータス領域に確認メッセージが表示されます。次に例を示します。

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

> **関連項目：** Rule Author セッションのタイムアウトの構成に関する詳細と、タイムアウトが発生した場合に Rule Author が現在の作業をディクショナリ・バージョンに自動的に保存する方法の詳細は、A-3 ページの「Rule Author のセッション・タイムアウト」を参照してください。

4.4 XML 版レンタカー・サンプルに対するデータ・モデルの定義

ルールでの作業前に、データ・モデルを定義する必要があります。データ・モデルには、Java クラス・ファクト・タイプ、XML ファクト・タイプおよび RL Language ファクト・タイプなど、ルールで使用されるファクトやデータ・オブジェクトに関するビジネス・データ定義が格納されます。ここでは、XML ファクト・タイプが格納されているデータ・モデルのみを取り扱います。

内容は次のとおりです。

- XML 版レンタカー・サンプルでのファクトとしての XML スキーマの使用
- レンタカー・サンプルに対する XML ファクトの追加（XML スキーマ処理）
- データ・モデルへの XML スキーマ要素のインポート
- データ・モデルでの XML ファクトの表示
- 現状の XML ファクト定義の保存

関連項目： 2-9 ページ「Rule Author への Java クラスおよびパッケージの追加」

4.4.1 XML 版レンタカー・サンプルでのファクトとしての XML スキーマの使用

XML サンプルには、*$HowToDir*/data ディレクトリの carrental.xsd ファイルが格納されています。このファイルは、ファクトをアサートするために XML 文書を使用する XML 版レンタカー・サンプルのスキーマを指定します。スキーマ定義にアクセスするには、*$HowToDir* を、XML How-To をインストールしたディレクトリに置換します。

4.4.2 レンタカー・サンプルに対する XML ファクトの追加（XML スキーマ処理）

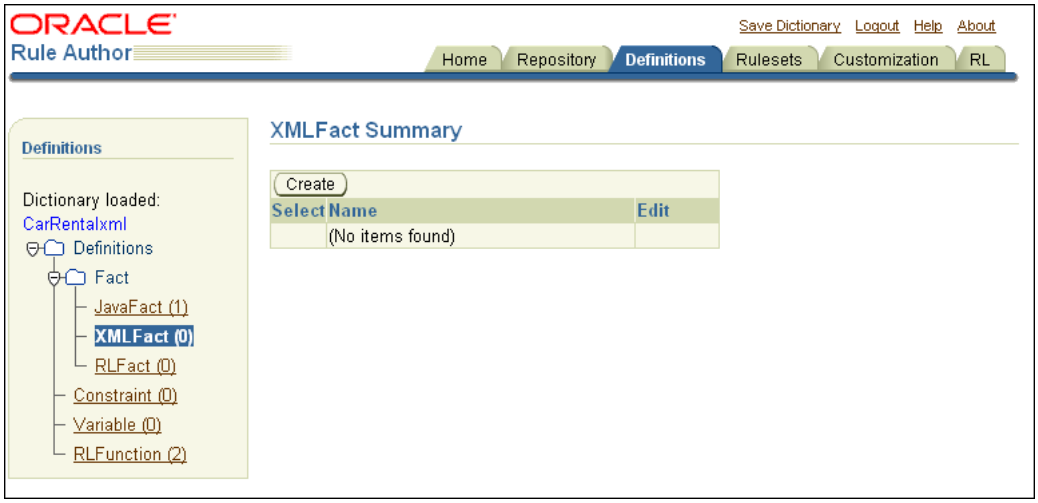
データ・モデルで XML 要素を使用する前に、Rule Author を使用して、XML 要素を表す JAXB クラスを生成する必要があります。ここでは、JAXB クラスを生成し、XML スキーマに関連付けられている生成したクラスとパッケージを Rule Author で参照できるようにします。

Rule Author を使用して XML 版レンタカー・サンプル・スキーマを準備する手順は、次のとおりです。

1. CarRentalxml ディクショナリを作成した直後の場合は、2 へ進んでください。**「リポジトリ」**タブをクリックし、CarRentalxml ディクショナリをロードします。
2. **「定義」**タブをクリックします。使用可能な定義が格納された「定義」フォルダがナビゲーション・ツリーに表示されます。
3. ツリーの「定義」フォルダには「ファクト」フォルダが表示されます。この「ファクト」フォルダには、使用可能なファクト・タイプ（**Java ファクト**、**XML ファクト**および **RL ファクト**）が含まれています。

   「XML ファクト」をクリックして「XML ファクト・サマリー」ページを表示します（図 4-4 を参照）。

図 4-4 Rule Author の「定義」タブにある「XML ファクト・サマリー」ページ



4. 「作成」をクリックします。「XML スキーマ・セレクタ」ページが表示されます。
5. 「XML スキーマ・セレクタ」ページで、「**XML スキーマ**」フィールドにスキーマに対するパスまたは HTTP URL を入力します。次に例を示します。
   - *$HowToDir*`/data/carrental.xsd`。*$HowToDir* には、XML 版 How-To をインストールしたディレクトリが入ります。
   - `http://www.myCompany.com/xsd/product.xsd`

   注意： URL を使用したスキーマへのアクセスを選択する場合で、プロキシ・サーバーが関係する場合は、次のシステム・プロパティを設定する必要があります。

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

   次に例を示します。

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

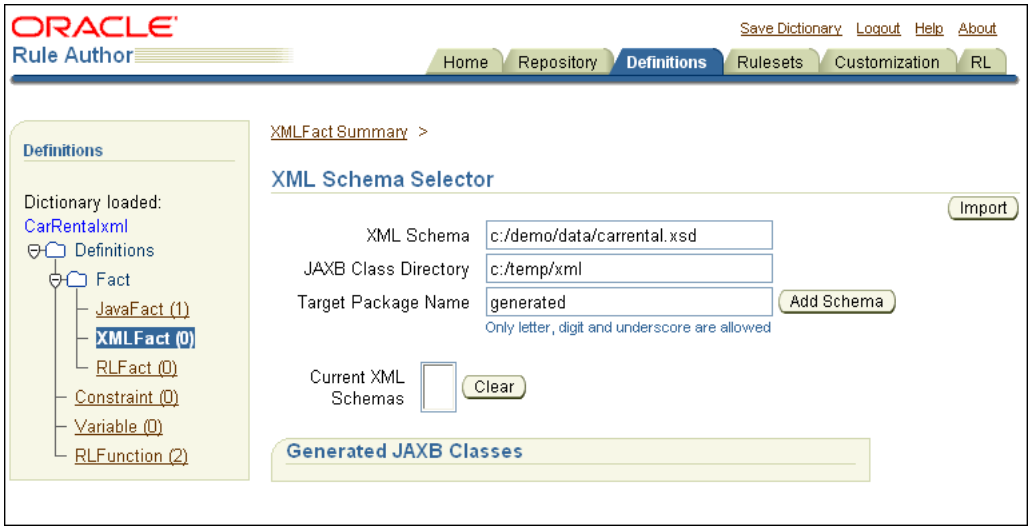
   OC4J インスタンスでのシステム・プロパティの設定の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

   ---

6. 「**JAXB クラス・ディレクトリ**」フィールドに、Rule Author で JAXB 生成クラスを格納するディレクトリを入力します。書込み可能なディレクトリを指定する必要があります。たとえば、`c:/temp/xml` と入力します。
7. 「**ターゲット・パッケージ名**」フィールドに値を入力します。このフィールドを空のままにすると、JAXB クラス・パッケージ名は、JAXB による XML から Java へのデフォルトのマッピング・ルールを使用して、XML スキーマの XML ターゲット名前空間から生成されます。たとえば、名前空間 rules.oracle.com は com.oracle.rules にマップされます。

   入力した値は、生成されるクラス・パッケージ名になります。たとえば、`generated` という名前を使用します（図 4-5 を参照）。この例では、`generated` という名前を使用しますが、この名前に特別な意味はありません。この名前で、生成されたクラスのパッケージが指定されます。

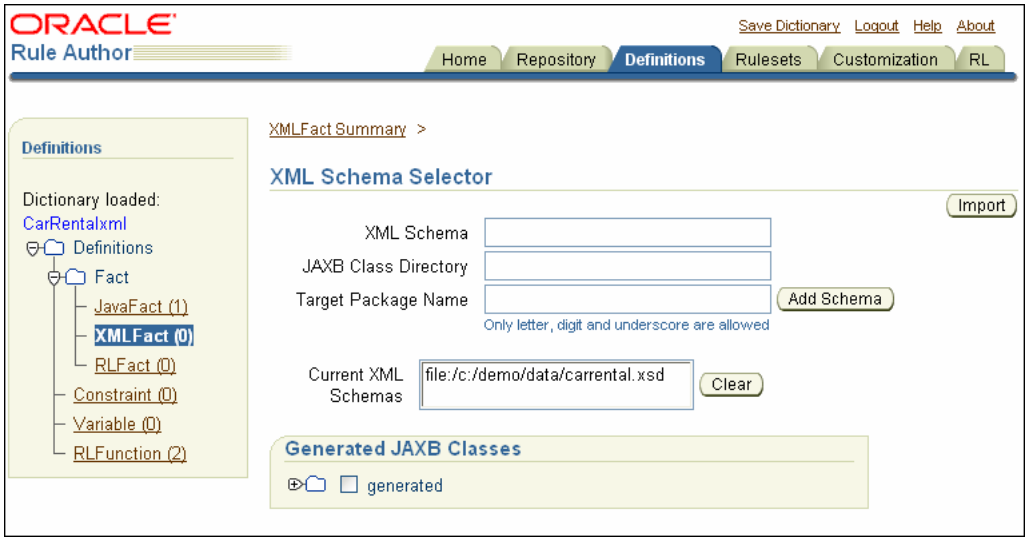
図 4-5 Rule Author の XML スキーマ選択ページ



8. **「スキーマの追加」**をクリックします。

   この手順では、Rule Author がスキーマを処理し、JAXB をコンパイルするのにしばらくの時間を要するため、スキーマのサイズによっては、この手順が完了するまで待機する必要があります。この手順が完了すると「スキーマの追加」テキスト入力フィールドが空白になり、Rule Author によって**「現在の XML スキーマ」**フィールドが更新され、「生成した JAXB クラス」領域が表示されます（図 4-6 を参照）。

図 4-6 Rule Author の「定義」タグにある「XML スキーマ・セレクタ」（XML スキーマ追加後）



注意： JAXB は、XML 構成名を異なる Java 識別名にマップすることがあります。たとえば、JAXB 生成クラスでは、`my-element-name` という XML 名は `myElementName` になります。Rule Author によって XML 構成名が提供されるため、JAXB で生成される XML から Java への名前マッピングを理解する必要がなくなります。

関連項目： 4-3 ページ「XML 版レンタカー・サンプルに対するディクショナリの作成および保存」

4.4.3 データ・モデルへの XML スキーマ要素のインポート

ここでは、XML スキーマ要素を表す JAXB 生成クラスを（サンプル・スキーマ `carrental.xsd` から）データ・モデルに移動します。**「定義」**タブから「XML スキーマ・セレクタ」ページを使用して、データ・モデルにインポートする XML 要素を選択します。

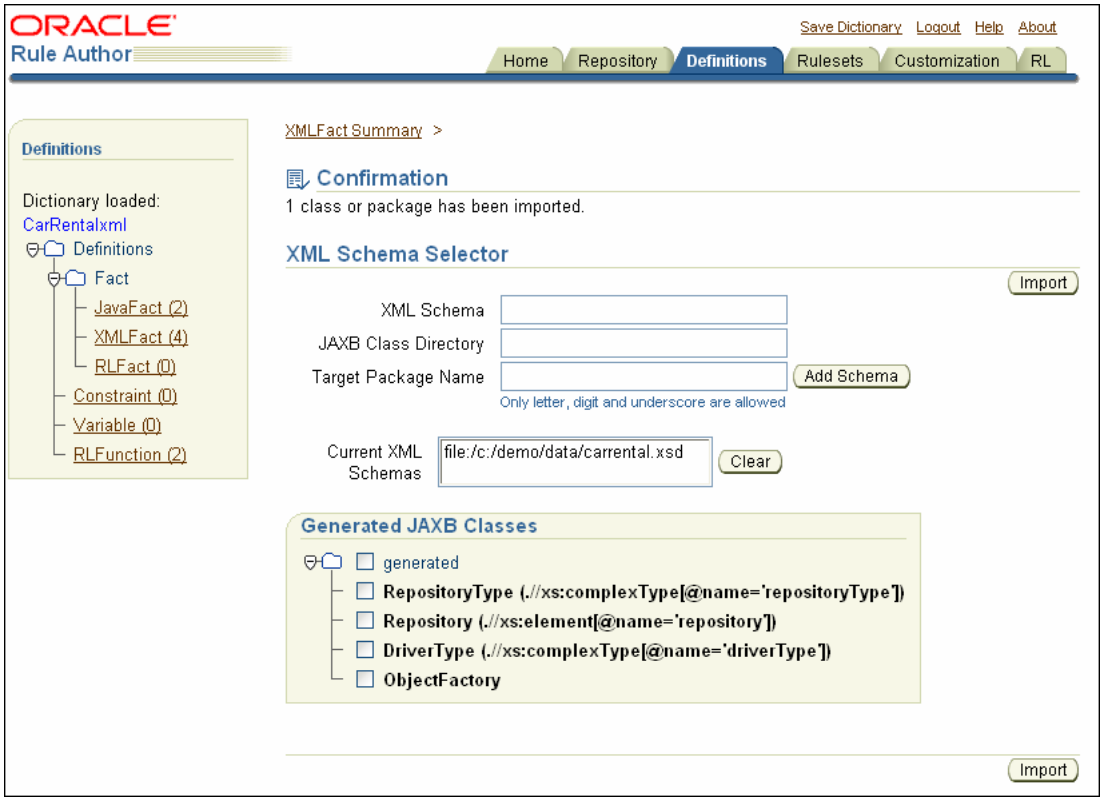
`DriverType` をスキーマからデータ・モデルに追加する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで「XML ファクト」フォルダをクリックします。
3. 「XML ファクト・サマリー」ページで**「作成」**をクリックします。「XML スキーマ・セレクタ」ページが表示されます。
4. 「XML スキーマ・セレクタ」ページの**「生成した JAXB クラス」**ボックスで、**「DriverType」**が表示されるまでナビゲーション・ツリーを開きます。
5. generated フォルダのチェック・ボックスを選択します。
6. **「インポート」**をクリックします。

   「1 つのクラスまたはパッケージがインポートされました。」という確認メッセージが表示されます。
7. 「生成した JAXB クラス」領域の **generated** ノードを開き、インポートされたクラスを確認します（図 4-7 を参照）。

注意： インポートされた要素は太字で表示されます。

図 4-7 Rule Author の「XML スキーマ・セレクタ」の確認メッセージ



XML スキーマを Rule Author に追加する際の注意点は、次のとおりです。

- Rule Author では、XML ファクトをインポートすることは、Java の import 文と同じことを意味します。つまり、Rule Author で JAXB クラスとそのメソッドを参照できるようになります。Rule Author では、クラスをデータ・モデルまたはディクショナリにコピーしません。
- 「クラス」ナビゲーション・ツリーは、要求に応じてレンダリングされます（パフォーマンス向上のため）。したがって、子ノードは親ノードを展開した場合にのみレンダリングされます。必要なノードのみを展開することをお薦めします。
- Windows システムでは、円記号（¥）またはスラッシュ（/）をパス・セパレータとして使用できます。Rule Author では、どちらのパス・セパレータも使用できます。
- 各 Java クラスの横には、Java クラスの生成元（XML スキーマ名を使用）を把握できるように、対応する XML 構成名が表示されます。パッケージ全体をデータ・モデルにインポートするには、パッケージ名を選択して**「インポート」**をクリックします。
- Java パッケージ名には、RL Language 予約語を使用しないでください。詳細は、D.8 項「Java パッケージ名の一部としての RL 予約語の使用」を参照してください。
- Oracle Business Rules では、JAXB バインディングを使用して XML スキーマを Java クラスにバインドします。ほとんどの場合は、Oracle JAXB バインディング・コンパイラで生成したデフォルトのバインディングでニーズを十分に満たせるはずです。ただし、デフォルト・バインディングの変更が必要になる場合があります。次に例を示します。
  - デフォルト・バインディングによって名前の重複が発生する場合。
  - デフォルト・バインディングによって無効な Java 識別子が生成される場合（英語以外のタグ名から無効な Java 識別子が生成される可能性があります）。

    関連項目： カスタム・バインディング宣言の使用に関する詳細は、Java Architecture for XML Binding（JAXB）仕様を参照してください。

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

4.4.4 データ・モデルでの XML ファクトの表示

JAXB 生成クラスまたはインポートしたパッケージを含めた XML ファクトをデータ・モデルに表示する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、「ファクト」フォルダを開き、**「XML ファクト」**ノードをクリックして「XML ファクト・サマリー」ページを表示します。

   XML 版レンタカー・サンプルでは、インポートされた DriverType クラス、RepositoryType クラス、Repository クラスおよび ObjectFactory クラスを含む XML ファクト表が表示されます。
3. 編集アイコンをクリックして、XML ファクトのプロパティとメソッドの表を表示します。

注意： Java クラスのインポートでは、そのすべてのスーパークラスと、フィールドおよびメソッドを介して関連付けられたクラスは、データ・モデルにインポートされません。これらのクラスに正しくアクセスするために、明示的にデータ・モデルにインポートする必要があります。

関連項目： XML ファクトのプロパティとメソッドの表の**「参照可能」**フィールドと**「拡張」**フィールドの詳細は、3-10 ページの「Rule Author のリストに関する参照可能性およびオブジェクト・チェーンの指定」を参照してください。

4.4.5 現状の XML ファクト定義の保存

「定義」タブでデータ・モデルを操作している場合、および作業を完了する場合は、ディクショナリを保存してください。

ディクショナリを保存する手順は、次のとおりです。

1. ページ上部の**「ディクショナリの保存」**リンクをクリックします。
2. 「ディクショナリの保存」ページで**「保存」**をクリックします。

4.5 XML 版レンタカー・サンプルに対するビジネス用語の定義

ビジネス用語を使用することで、ビジネス・アナリストは、XML 名や Java パッケージ名、クラス名、メソッド名またはメンバー変数名を使用せずに、簡単に理解できる名前を使用してルールを作成できます。Rule Author には別名機能が用意されています。この機能を使用すると、ビジネスマン向けの用語を使用してルールのビジネス・オブジェクトまたはファクトを参照できます。唯一必要な手順は、ルールに使用する予定のビジネス・オブジェクトに対してビジネス用語を定義することです。また、**「ルールセット」**タブからルールを作成する際は、Rule Author の**「参照可能」**ボックスを使用して、Rule Author のリストに表示するプロパティとメソッドを指定します。

内容は次のとおりです。

- XML ファクト定義に対するビジネス用語の指定
- 関数に対するビジネス用語の指定
- XML ファクトのプロパティとメソッドに対する参照可能性の指定

4.5.1 XML ファクト定義に対するビジネス用語の指定

XML ファクト定義に対してビジネス用語を指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、「ファクト」フォルダを開き、**「XML ファクト」**ノードをクリックして「XML ファクト・サマリー」ページを表示します。XML 版レンタカー・サンプルでは、`generated.DriverType` クラスを含む表が表示されます（generated 以外のパッケージ名を指定した場合、パッケージ名は `generated` とは異なります）。
3. `DriverType` の編集アイコンをクリックします。「XML ファクト」ページが表示されます。
4. 「XML ファクト」ページ上部の**「エイリアス」**フィールドに `DriverData` と入力します。
5. 「プロパティ」表の `age` エントリに対して、適切な別名を指定します。たとえば、**「エイリアス」**フィールドに `DriverAge` と入力します。
6. 「プロパティ」表の `name` エントリに対して、適切な別名を指定します。たとえば、**「エイリアス」**フィールドに `DriverName` と入力します。
7. **「OK」**または**「適用」**をクリックします。
8. ディクショナリを保存します。

注意： 変更後は必ず**「OK」**または**「適用」**をクリックしてください。いずれかのボタンをクリックしないと、Rule Author では変更内容が保存されません。

XML ファクト定義に対してビジネス用語を指定する際の注意点は、次のとおりです。

- 「XML ファクト」ページには、Java クラスが XML スキーマから生成されたことを示す**「XML 名」**フィールドと**「生成元」**フィールドがあります。

  たとえば、`//complexType[@name='driverType']` のように表示されます。**「XML 名」**は、クラスが `driverType` という名前の XML 複合型から生成されたことを示しています。**「生成元」**フィールドは、XML ファクトに対して JAXB クラスを生成した XML スキーマの名前を示します。

- 「XML ファクト」ページでは、スーパークラス・チェーン内の指定したメソッドまたはプロパティの、1 レベル上のメソッドまたはプロパティが表示されるように指定できます。そのためには、「XML ファクト」ページで、そのメソッドまたはプロパティの**「拡張」**ボックスを選択します。この「拡張」ボックスは、「プロパティ」領域または「メソッド」領域の**「拡張」**フィールドに表示されます。チェック・ボックスは、スーパークラスがあるメソッドまたはプロパティに対してのみ表示されます（Rule Author では、プリミティブ型には**「拡張」**ボックスは表示されません）。

- 「XML ファクト」ページでは、Rule Author のリストにプロパティまたはクラスが表示されないように指定できます。オブジェクトが Rule Author のリストに表示されないように指定するには、**「参照可能」**チェック・ボックスの選択を解除します（デフォルトでは、オブジェクトはリストで参照可能です）。

- すべての XML ファクト・タイプに対して**「XPath アサーションのサポート」**ボックスが選択されていることを確認してください。詳細は、D.10 項「実行時にアサートされない XML ファクト」を参照してください。

4.5.2 関数に対するビジネス用語の指定

RL Language 関数に対してビジネス用語を指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、「定義」フォルダの「RL 関数」をクリックし、「RL 関数サマリー」ページを表示します。XML 版レンタカー・サンプルでは、`DM.assertXPath` 関数および `DM.println` 関数を含む表が表示されます。
3. `DM.println` 関数について、**「編集」**フィールドの編集アイコンをクリックして詳細を表示します。
4. **「名前」**フィールドの下にある**「エイリアス」**フィールドに別名を入力します。たとえば、**「エイリアス」**フィールドに `PrintOutput` と入力します。

   注意：「エイリアス」フィールドは「関数の引数」表にもあります。この例のような場合は、「関数の引数」の「エイリアス」フィールドを変更しないでください。

5. **「OK」**または**「適用」**をクリックします。
6. ディクショナリを保存します。

4.5.3 XML ファクトのプロパティとメソッドに対する参照可能性の指定

プロパティまたはメソッドが Rule Author のリストで参照可能かどうかを指定する手順は、次のとおりです。

1. **「定義」**タブをクリックして「定義」ページを表示します。
2. ナビゲーション・ツリーで、**「XML ファクト」**ノードをクリックして「XML ファクト・サマリー」ページを表示します。
3. 編集アイコンをクリックして、`DriverType` に対する XML ファクトのプロパティとメソッドを表示します（表では、このエントリの値は「エイリアス」列の `DriverData` です）。「XML ファクト」ページが表示されます。
4. 「プロパティ」表と「メソッド」表の各エントリについて、**「参照可能」**チェック・ボックスを使用して適切な参照可能性を指定します。この例では、メンバー変数 `age` および `name` のみを参照可能にする必要があります。
5. **「OK」**または「適用」をクリックして変更内容を保存します。
6. ディクショナリを保存します。

注意： 特定のプロパティまたはメソッドに対する参照可能インジケータを変更すると、依存する定義またはルールが誤って表示されることがあります。この状況が発生する場合は、問題の原因となっている参照不可のプロパティまたはメソッドを参照可能に設定してください。

4.6 XML 版レンタカー・サンプルに対するルールの定義

ここでは、XML 版レンタカー・サンプルのルールを定義します。

内容は次のとおりです。

- XML 版レンタカー・サンプルに対するルールセットの作成
- XML 版レンタカー・サンプルに対するルールの作成

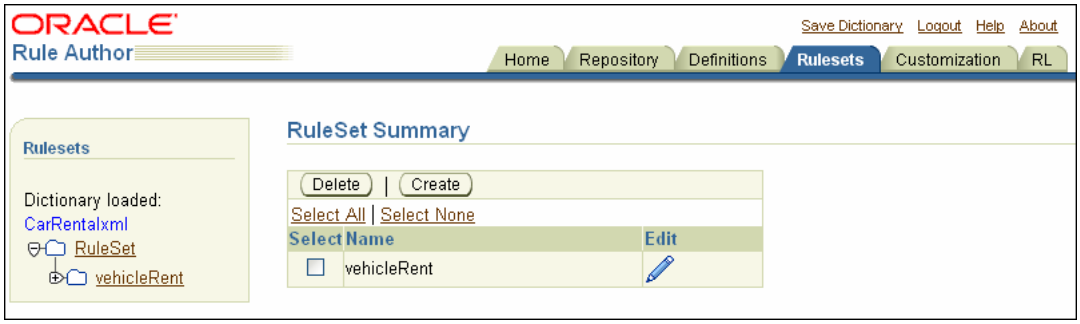
4.6.1 XML 版レンタカー・サンプルに対するルールセットの作成

ルールを作成する前に、ルールセットを作成する必要があります。ルールセットとはルールのコンテナです。

ルールセットを作成する手順は、次のとおりです。

1. **「ルールセット」**タブをクリックします。
2. ナビゲーション・ツリーで**「ルールセット」**ノードをクリックします。
3. 「ルールセット・サマリー」ページで、**「作成」**をクリックします。「ルールセット」ページが表示されます。
4. **「名前」**フィールドに名前を入力します。たとえば、`vehicleRent` と入力します。
5. （オプション）**「説明」**フィールドにテキストを入力します。たとえば、`vehicle rent rule set` と入力します。
6. **「OK」**をクリックします。`vehicleRent` ルールセットが作成されます。ルールセットを作成すると、図 4-8 のように新しいエントリがツリーに表示されます。
7. ディクショナリを保存します。

図 4-8 Rule Author の「ルールセット・サマリー」ページ



4.6.2 XML 版レンタカー・サンプルに対するルールの作成

ルールセットを作成した後は、ルールセットにルールを作成できます。ここでは、UnderAge ルールを作成します。UnderAge ルールは、次の内容をテストします。

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

UnderAge ルールには、Rules Engine が照合する単一のパターンと、そのパターンに適用されるテストが指定されます。

次のアクションが UnderAge ルールに関連付けられます。

- テキスト「Rental declined」、一致した運転者の名前、メッセージ「Under Age, age is: 」、および運転者の年齢の出力。
- ルール・セッションからの一致した運転者ファクトの取消し。様々な理由から、ファクトの取消しが必要になることがあります。たとえば、ファクトの使用が終了して、Rules Engine から削除する場合や、ルールに関連付けられているアクションによって状態が変化したため、Rules Engine の現在の状態が示されるように、ファクトを取り消す必要がある場合などです。

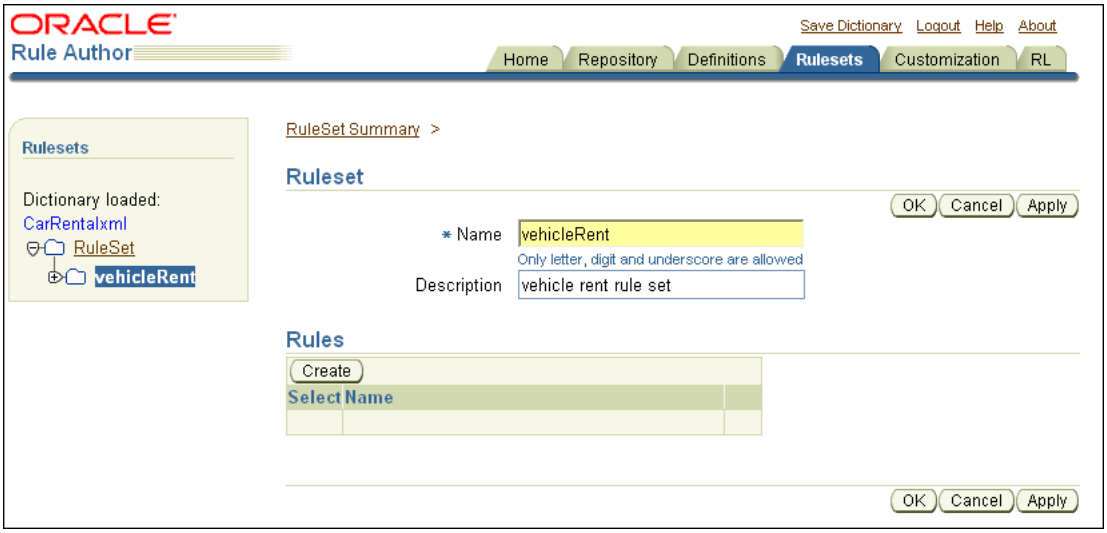
4.6.2.1 XML 版レンタカー・サンプルに対する UnderAge ルールの追加

Rule Author を使用して UnderAge ルールを追加する手順は、次のとおりです。

1. 「**ルールセット**」タブをクリックします。ナビゲーション・ペインには、4.6.1 項で作成した vehicleRent ルールセットが格納された「ルールセット」フォルダが表示されます。
2. ナビゲーション・ツリーで、**vehicleRent** ノードをクリックします。「ルールセット」ページが表示され、ルールが表にリストされます（図 4-9 を参照）。

注意： ルールがない場合、「ルール」表は空です。

図 4-9「作成」ボタンが表示されている Rule Author の「ルールセット」ページ

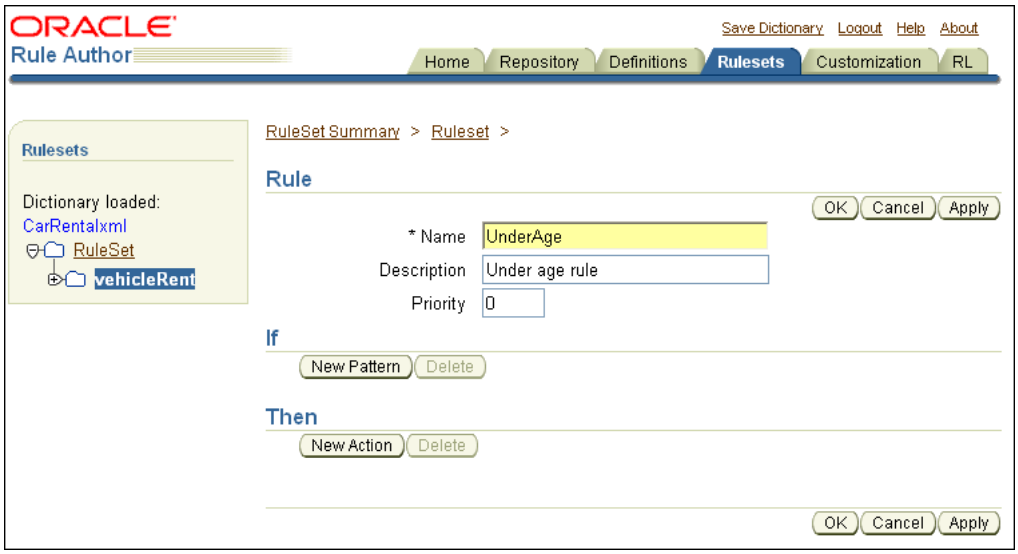


3. **「作成」**をクリックします。「ルール」ページが表示されます。
4. 「ルール」ページで、**「名前」**フィールドに `UnderAge` と入力します。
5. 「ルール」ページで、**「優先度」**フィールドに `0` を入力します。

> **注意：** **「優先度」**フィールドは、複数のルールを適用する場合に、どのルールをどのような順序で適用するかを決定します。ルールを使用するアプリケーションの多くでは、これが未決定でルールセット内のルールは順不同で適用され、また、優先度の設定も必須ではありません。

6. （オプション）**「説明」**フィールドに説明を入力します（図 4-10 を参照）。

図 4-10 Rule Author の「ルール」ページ



4.6.2.2 UnderAge ルールに対するパターンの追加（XML）

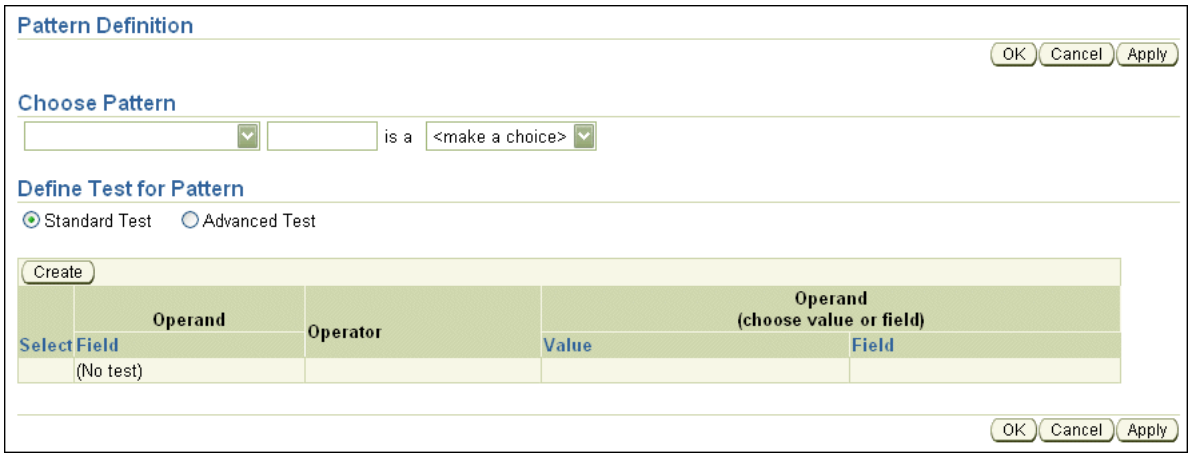
Rules Engine を実行すると、ルールを使用して、一致するパターンに使用可能なファクトがチェックされます。UnderAge ルールのパターンを追加する手順は、次のとおりです。

1. 「ルール」ページで、**「if」**ボックスの**「新規パターン」**をクリックします。「パターン定義」ページが表示されます。

   「パターン定義」ページには、「パターンの選択」と「パターンのテストを定義」の 2 つの領域があります（図 4-11 を参照）。

   > **注意：**「パターン定義」ページが表示されない場合は、ブラウザのポップアップ遮断機能が使用可能になっている可能性があります。Rule Author を使用するには、ポップアップ遮断機能を使用不可にする必要があります。

図 4-11 Rule Author の「パターン定義」ページ



2. **「パターンの選択」**の下にある最初のボックスで、最初のエントリ（空白）を選択します。

   このボックスは、一致するたびに（一致するすべての運転者に）、ルールを起動することを指定します。別の値の 1 つである「次のケースが少なくとも 1 つある」は、少なくとも 1 つが一致する（そのような運転者が 1 人いる）場合に、ルールを 1 回起動することを選択します。別の値の「次のケースがない」は、一致しない（一致する運転者がいない）場合に、ルールを 1 回起動することを指定します。

3. **「パターンの選択」**の下にある隣の領域には、一致したファクトに暫定的な名前を入力できます。

   このフィールドに `driver` と入力（パターンのバインド変数名を定義）します。

   このフィールドを使用して、単一のルールで同じタイプの複数のインスタンスをテストできます。たとえば、`driver1.age > driver2.age` などの比較で、指定した名前を使用して、ある運転者と他の運転者を比較できます。

4. 3 番目のボックスには、`<make a choice>` というテキストが表示されます。このボックスには、使用可能なファクト・タイプが表示されます。このボックスで、`DriverData` を選択します。

5. **「OK」**をクリックし、パターン定義を保存して「パターン定義」ページを閉じます。

6. 「ルール」ページで、**「OK」**または**「適用」**をクリックしてルールを保存します。

> **注意：** パターンの変更内容は、「ルール」ページで**「OK」**または**「適用」**をクリックするまで、ルールには追加されません。**「OK」**または**「適用」**をクリックせずに別のルールセットにナビゲートしたり、異なるタブを選択すると、パターン定義の変更内容は破棄されます。

7. ディクショナリを保存します。

パターンに対してテストが定義されていないと、定義するアクションはすべての運転者に適用されます。パターンにテストを定義するには、4.6.2.3 項の説明に従って操作を続行します。

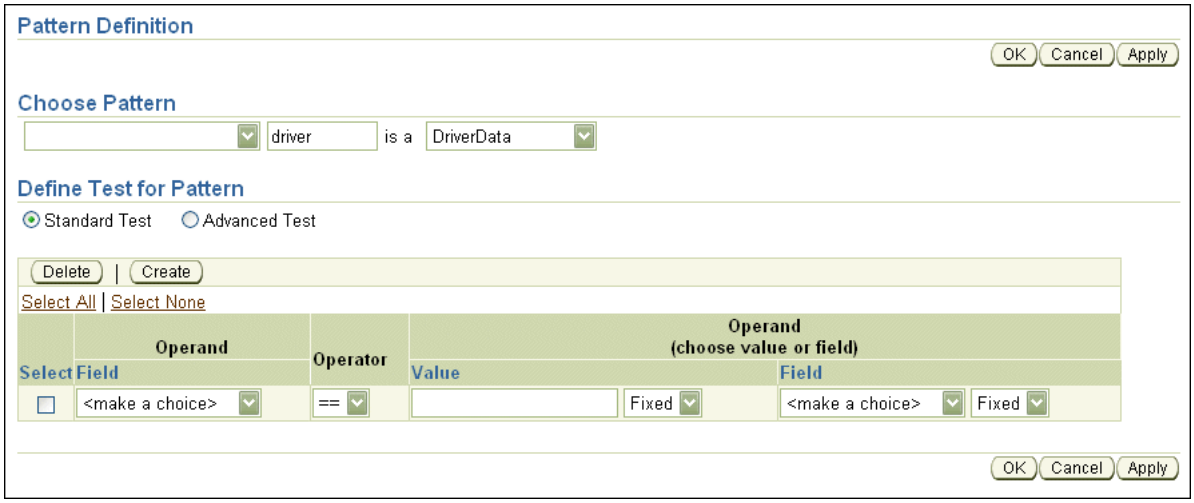
> **関連項目：** 4-20 ページ「UnderAge ルールに対するアクションの追加（XML）」

4.6.2.3 UnderAge ルールでのパターンに対するテストの定義（XML）

パターンに対してテストを追加する手順は、次のとおりです。

1. **「ルールセット」**タブから、テストを追加するルールをナビゲーション・ツリーでクリックします。この例では、UnderAge ルールのノードをクリックします。
2. 「ルール」ページの**「if」**表で、鉛筆アイコンを選択し、このルールの「パターン定義」ページを表示します。
3. 「パターン定義」ページで、**「標準テスト」**ボタンを選択し、次に**「作成」**をクリックします（図 4-12 を参照）。詳細は、3.8.1 項「「アドバンスト・テスト式」オプションの使用」を参照してください。

図 4-12 Rule Author のルールの「パターン定義」ページの「新規テスト」フィールド



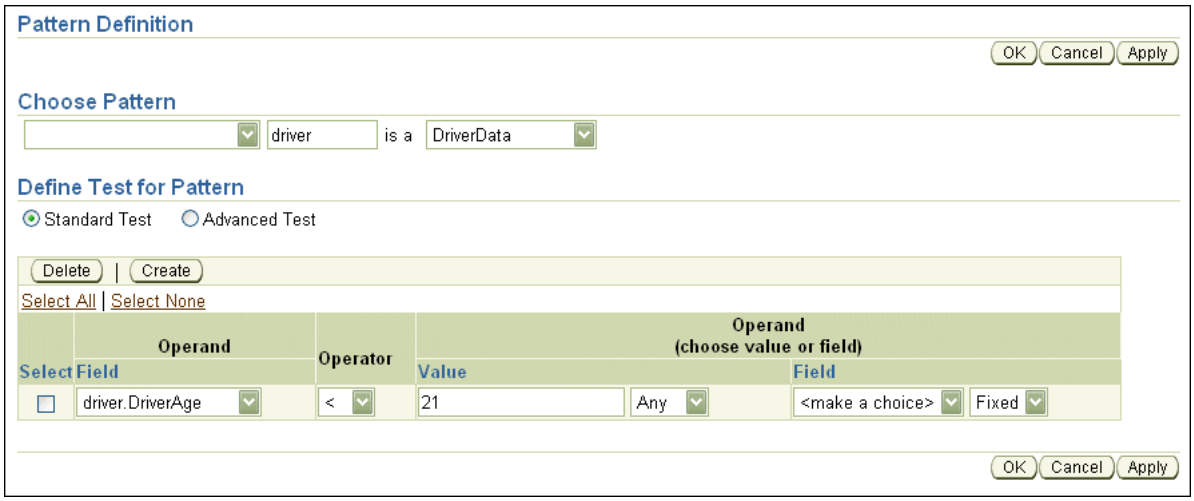
4. **「オペランド」**列で、**「フィールド」**ボックスから `driver.DriverAge` を選択します。
5. **「演算子」**列で、<（より小さい）を選択します。
6. 次の**「オペランド」**列で、**「値」**ボックスに 21 を入力します。**「フィールド」**ボックスには値を入力しません。

7. 各「**値**」ボックスと「**フィールド**」ボックスの横には、Fixed と Any の値を備えたリストがあります（図 4-13 を参照）。

   「**値**」フィールドの制約として Any を選択します。

   これらの値は制約と呼ばれます。制約値を使用してカスタマイズを有効化または無効化します。Fixed 値を使用して、フィールドを読取り専用にします。これは、この値に対してカスタマイズを禁止することを指定します。Any 値を選択すると、Rule Author での値の変更が許可されることを指定します。Any の値を設定すると、ルールをカスタマイズできます（専門外のユーザーによる変更がサポートされます）。許可する値を制限する制約も定義できます。

図 4-13 年齢制限ルールの値が表示された Rule Author の「パターン定義」ページ



8. 「**OK**」をクリックし、変更内容を保存して「パターン定義」ページを閉じます。
9. 「ルール」ページで、「**OK**」または「**適用**」をクリックします。

> **注意：** パターンの変更内容は、「ルール」ページで「**OK**」または「**適用**」をクリックするまで、ルールには追加されません。「**OK**」または「**適用**」をクリックしないと、ルールに対する変更内容は保存されません

10. ディクショナリを保存します。

パターンに対してテストを追加する際の注意点は、次のとおりです。

- **標準テスト**式では、テストが true と評価されるのは、定義するすべてのテストが一致した場合のみです。また、標準テストでは、グループ化や、パラメータを使用する関数は許可されません。ただし、標準テストを使用すると、カスタマイズに対して制約を定義できます。
- **アドバンスト・テスト**式では、テストに標準テストのような制限はありませんが、制約は使用できません。アドバンスト・テスト式は、RL Language を使用して直接保存されることはありません。Rule Author によって式に別名が取り込まれるためです。RL Language では、別名はサポートされていません（Rule Author で別名は変数名にマップされます）。

> **関連項目：** 4-23 ページ「XML 版レンタカー・サンプルに対するルールのカスタマイズ」

4.6.2.4 UnderAge ルールに対するアクションの追加（XML）

アクションはパターン一致と関連しています。実行時に、ルールの「if」部が一致すると、ルールに関連付けられているアクションを実行するために、Rules Engine によって「then」部が実行されます。

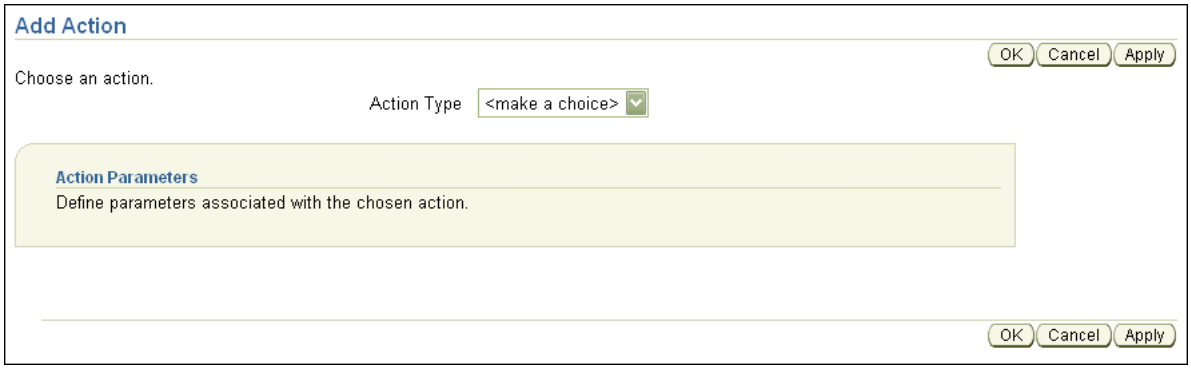
ここでは、UnderAge ルールに対して 2 つのアクションを追加します。第 1 のアクションは結果の出力です。第 2 のアクションは、Rules Engine からの運転者ファクトの取消しです。次のような様々な理由で、ファクトの取消しが必要になることがあります。

- ファクトの使用が終了し、Rules Engine から削除する場合。
- ルールに関連付けられているアクションによって状態が変化したため、Rules Engine の現在の状態が示されるように、ファクトを取り消す必要がある場合。

UnderAge ルールの一致に関する結果を出力するアクションを追加する手順は、次のとおりです。

1. **「ルールセット」**タブをクリックします。
2. ツリーで、vehicleRent フォルダの下の UnderAge ルールのノードをクリックします。
3. 「ルール」ページで、**「then」**ボックスの**「新規アクション」**をクリックします。「アクションの追加」ページが表示されます（図 4-14 を参照）。

図 4-14 Rule Author の「アクションの追加」ページ



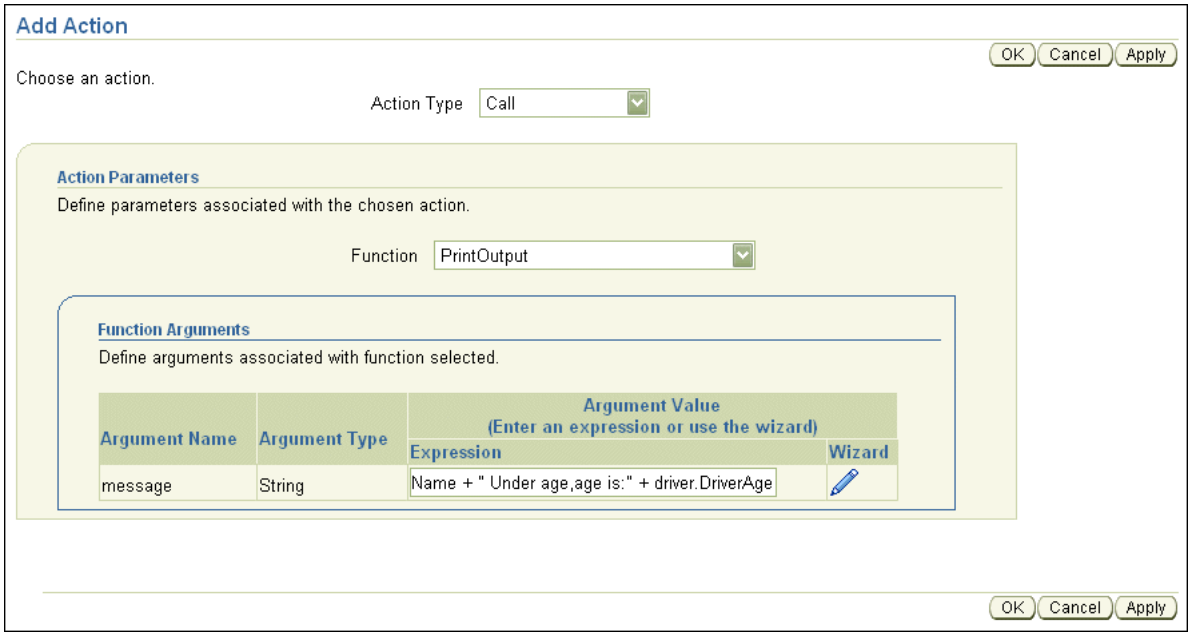
4. **「アクション・タイプ」**ボックスから `Call` 項目を選択します。**「アクション・パラメータ」**ボックスが表示されます。
5. **「関数」**ボックスから `PrintOutput`（`println` の別名を定義していない場合は `DM.println`）を選択します。**「関数の引数」**ボックスが表示されます。
6. **「引数の値」**フィールドに引数の値を入力します（図 4-15 を参照）。

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

注意 1: Rule Author では、Java に類似した構文を式に対して使用します。RL Language では、完全な式の構文を定義します。

注意 2: 「ウィザード」フィールド内で編集アイコンを選択し、ウィザードを使用して式を入力することもできます。このウィザードでは、式を記述するための追加領域が提供されます。また、ウィザードには使用可能な変数のリストも用意されているため、変数をより簡単かつ正確に入力することができます。

図 4-15 UnderAge ルールに関する Rule Author の「アクションの追加」ページ

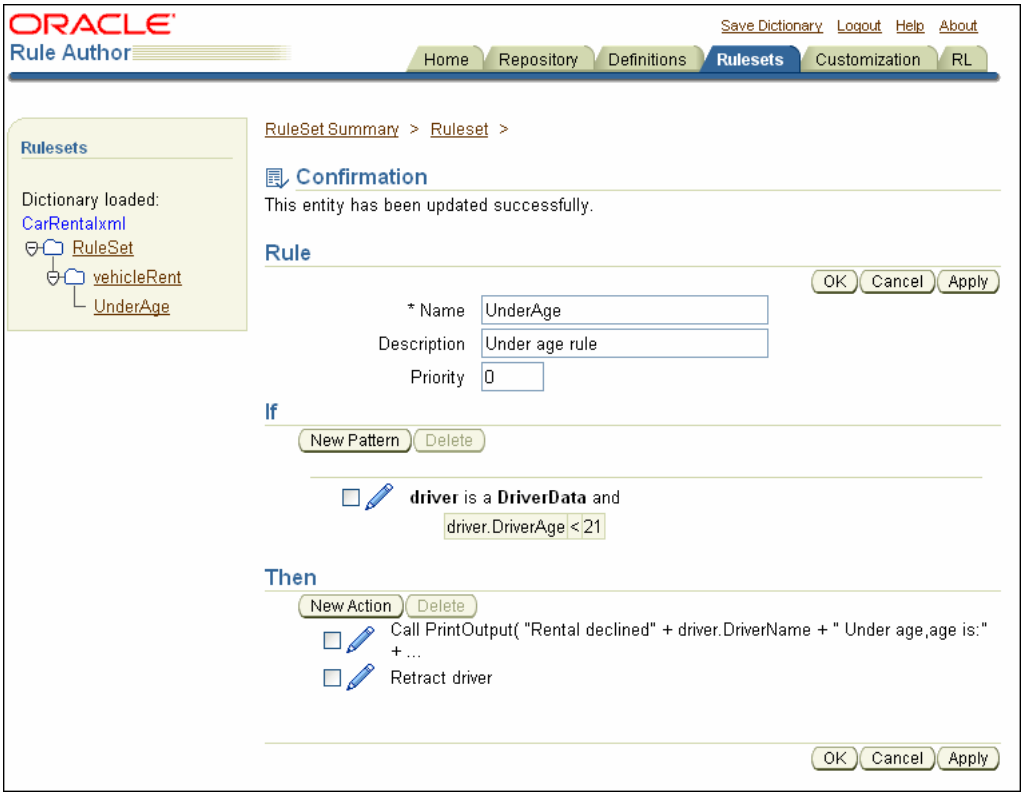


7. **「OK」**をクリックし、変更内容を保存して「アクションの追加」ページを閉じます。
8. 「ルール」ページで、**「OK」**または**「適用」**をクリックします。
9. 変更内容を保存するためにディクショナリを保存します。

次に、UnderAge ルールに対して取消しアクションを追加します。次の手順を実行し、この第 2 のアクションをルールに追加します。

1. **「ルールセット」**タブをクリックします。
2. vehicleRent フォルダの下の UnderAge ルールのノードをクリックします。
3. 「ルール」ページで、**「then」**ボックスから**「新規アクション」**をクリックします。「アクションの追加」ページが表示されます。
4. **「アクション・タイプ」**ボックスから `Retract` を選択します。**「アクション・パラメータ」**ボックスが表示されます。
5. **「ファクト・インスタンス」**ボックスから `driver` を選択します。アクションで使用される場合、パターン名（`driver`）は、パターンが一致した単一のインスタンスを参照します。
6. **「OK」**をクリックし、変更内容を保存して「アクションの追加」ページを閉じます。
7. 「ルール」ページで、**「OK」**または**「適用」**をクリックして変更内容を保存します（図 4-16 を参照）。
8. ディクショナリを保存します。

図 4-16 Rule Author の年齢制限ルールのパターンとアクション



注意： アクションをルールに追加する場合、連続して追加できるのは新しいアクションのみです。あるアクションがそれ以前のアクションの結果に依存する場合は、アクションを追加する順序が重要になります。

関連項目： 『Oracle Business Rules ランゲージ・リファレンス』

4.7 XML 版レンタカー・サンプルに対するルールのカスタマイズ

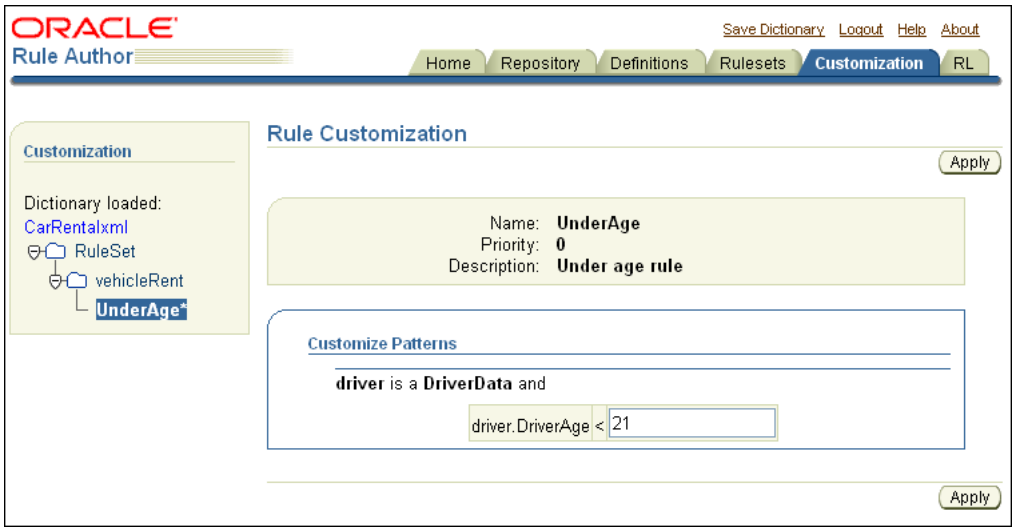
Rule Author の**「カスタマイズ」**タブは、ビジネス・ユーザー用に設計されています。ルール開発者は、**「ルールセット」**タブから使用できる「パターン定義」ページの**許可された値**のフィールドを使用して、カスタマイズを許可するかどうかを指定します。カスタマイズが許可されている場合は、カスタマイズ可能な値について、値の範囲を指定できます。その結果、ビジネス・ユーザーは**「カスタマイズ」**タブを使用して値を変更できます。

この例では、**「カスタマイズ」**タブで、運転者の年齢制限を変更するように UnderAge ルールを変更できます（このサンプルでは、値に制限はなく、あらゆる指定値が有効です）。

「カスタマイズ」タブを使用して UnderAge ルールを変更する手順は、次のとおりです。

1. **「カスタマイズ」**タブをクリックします。ナビゲーション・ペインには vehicleRent フォルダが表示され、その下に、後ろにアスタリスク（*）の付いた UnderAge ルールのノードが表示されています。このアスタリスクは、ルールがカスタマイズ可能なことを示しています。
2. UnderAge ルールのノードをクリックします（図 4-17 を参照）。

図 4-17 年齢制限ルールに関する Rule Author の「ルールのカスタマイズ」ページ



3. 「ルールのカスタマイズ」ページの「パターンのカスタマイズ」ボックスには、`driver.DriverAge < 21` のテストを編集できるテキスト入力フィールドがあります。

   このフィールドに 19 を入力（21 から 19 に値を変更）します。
4. **「適用」**をクリックします。
5. ディクショナリを保存します。

ディクショナリを保存すると、XML 版レンタカー・サンプルに対するデータ・モデルとルールの作成は終了となります。

関連項目： 4-18 ページ「UnderAge ルールでのパターンに対するテストの定義（XML）」

4.8 XML ファクトを使用したルール・セッションがある Java アプリケーションの作成

データ・モデルおよびルールを指定したルールセットが格納されている Rule Author ディクショナリを作成および保存した後は、既存の Java アプリケーションをルール対応にしたり、新しいルール対応 Java アプリケーションを作成できます。この項では、ルール対応アプリケーションを作成する手順を示します。

内容は次のとおりです。

- Rules SDK および Rules RL クラスのインポート
- JAXB コンテキストの作成と XML 文書のアンマーシャリング
- Rules SDK でのリポジトリの初期化
- Rules SDK でのディクショナリのロード
- ルールセットのロードおよびデータ・モデルとルールセットに対する RL Language の生成
- ルール・セッションの初期化および実行
- ルール・セッション内からの XML データのアサート
- ルール・セッションでの実行関数の使用

このサンプル・アプリケーションの完全なコードは、$*HowToDir*/src/carrental ディレクトリの TestXML.java を参照してください。$*HowToDir* には、XML How-To をインストールしたディレクトリが入ります。

注意 1: 第 2 章で Java 版のレンタカーの例を完了している場合、この例での違いは、JAXB コンテキストを作成する必要がある点と、ファクトをルール・セッションに追加するために assertXPath 関数を使用する必要がある点です。

注意 2: この章に前述されている項の説明に従うことで、WebDAV リポジトリおよび CarRentalxml というディクショナリが作成および保存されています。How-To サンプル・コードで提供されているレンタカーの例でも、CarRentalxml という名前のディクショナリが格納されているファイル・リポジトリを使用します。この章で作成した WebDAV リポジトリと、How-To サンプルのファイル・リポジトリのディクショナリ内容は同じです。

How-To サンプルのコードには、WebDAV リポジトリとファイル・リポジトリの両方のコードが含まれていますが、ファイル・リポジトリのみが詳細に説明されています。How-To サンプルでは、移植性の観点からファイル・リポジトリを使用していますが、前述の項で作成した WebDAV リポジトリを使用するように変更できます。

4.8.1 Rules SDK および Rules RL クラスのインポート

ルール対応プログラムを記述する最初の手順は、特定の必要なクラスのインポートです。例 4-1 に、XML 版レンタカー・サンプルの TestXML.java アプリケーションからのインポートを示します。

例 4-1 Rules SDK での XML 版レンタカー・サンプルに必要なインポート

```
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 JAXB コンテキストの作成と XML 文書のアンマーシャリング

JAXB 生成クラスを使用して、最初に JAXB コンテキストを指定し、次にスキーマに準拠する XML 文書をアンマーシャリングする必要があります。例 4-2 に、このコードを TestXML.java から示します。

例 4-2 XML 文書のアンマーシャリング

```
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 Rules SDK でのリポジトリの初期化

ルール対応 Java アプリケーションを作成する場合は、次の手順を実行し、ディクショナリにアクセスしてルールセットを指定します（例 4-3 を参照）。

1. リポジトリのパスが指定されている String を作成します。
2. Rules SDK の RuleType オブジェクトを使用して、RepositoryManager.getRegisteredRepositoryType メソッドから取得するリポジトリを保持します。
3. リポジトリ・マネージャの createRuleRepositoryInstance メソッドを使用してリポジトリ・インスタンスを作成します。
4. RepositoryContext を定義し、適切なプロパティを設定します。ファイル・リポジトリの場合は、repoPath パラメータに伴って記載されているように、リポジトリのパスを指定します。
5. RuleRepository オブジェクト repo の init メソッドを使用して、リポジトリ・インスタンスを初期化します。

例 4-3 Rules SDK でのディクショナリのロード（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 );
```

例 4-3 のように、ファイル・リポジトリではなく WebDAV リポジトリをロードするには、`getWebDAVRepository` を使用してください。この例は、$*HowToDir*`/src/carrental` ディクショナリの `TestXML.java` にあります。

4.8.4 Rules SDK でのディクショナリのロード

ルール対応 Java アプリケーションを作成する場合は、ディクショナリを指定バージョンでロードする必要があります。例 4-4 のように、`RuleDictionary` オブジェクトを使用してディクショナリをロードします。この例では、`CarRental` ディクショナリを `HowTo` バージョンで `dict` という名前のオブジェクトにロードします。`CarRental` ディクショナリは、リポジトリで使用可能であることが必要です（前述の項で Rule Author を使用して、バージョン名が `HowTo` の `CarRental` ディクショナリを作成しました）。

例 4-4 Rules SDK でのディクショナリのロード

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

4.8.5 ルールセットのロードおよびデータ・モデルとルールセットに対する RL Language の生成

ディクショナリをロードした後は、Rules SDK を使用して RL Language プログラムを生成できます。ディクショナリは中間 XML 形式を使用してデータ・モデルとルールセットを格納するため、この手順は必須です。`RuleDictionary` オブジェクトは、データ・モデルとルールセットにアクセスするためのメソッドを提供し、中間 XML 形式からマッピングを実行します。このマッピングで、RL Language データ・プログラムが作成されます。

Rule Author を使用してルールを生成する場合は、各ルールセットが 2 つのコンポーネント（ディクショナリのルールセットすべてに適用されるグローバルなデータ・モデルと、ルールセットに関連付けられている一連のルール）を指定します。

例 4-5 に、ルールセットおよび関連するデータ・モデルについて RL Language コードを生成するコードを示します。

例 4-5 Oracle Business Rules RL Language の生成

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

4.8.6 ルール・セッションの初期化および実行

ルールとデータ・モデルが挿入されている RL Language プログラムを生成すると、いつでもルール・セッションで作業できる状態になります。ルール・セッションは、Rules Engine を初期化し、多数のルール実行にわたって Rules Engine の状態を保守します。

例 4-6 に、RuleSession オブジェクトを作成し、RL Language プログラムを実行するコードを示します。

executeRuleset() メソッドは、指定の RL Language プログラムの解析を Rules Engine に指示します。

注意： executeRuleset() コールの順序は重要です。ルールセットの RL Language プログラムを実行する前に、データ・モデルの RL Language プログラムを実行する必要があります。データ・モデルには、関連するルールセットを実行するときに必要なグローバル情報が格納されます。

例 4-6 Rules SDK でのルール・セッションの初期化および実行（XML）

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

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

データ・モデルとルールセットがロードされると、ルール・セッションは、そのルール・セッション用にアサートするファクトに対して、いつでもルールセットを実行できる状態になります。

4.8.7 ルール・セッション内からの XML データのアサート

ルール・セッションを実行する前に、XML データが格納されている XML 文書をアンマーシャリングし、次に XML 文書からファクトをアサートする必要があります。4.8.2 項に、XML 文書のアンマーシャリング方法を示します。

XML 文書からファクトをアサートするには、assertXPath 関数を引数に指定して session.callFunctionWithArgument() メソッドを使用します。

例 4-7 に、assertXPath を使用して XML ファクトをルール・セッションにアサートするサンプル・コードを示します。

callFunctionWithArgumentList メソッドには、関数名引数とリスト引数が必要です。リスト引数 argList には、次の 3 つの引数が含まれます。

1. assertXPath の第 1 引数は、JAXB で生成したパッケージ名です。この例では generated です。
2. assertXPath の第 2 引数は、アンマーシャリングされた XML 文書のルート・オブジェクトです。この例では、アンマーシャリングされたオブジェクト参照は root オブジェクトです。
3. assertXPath の第 3 引数は、アサートする XPath 式です。この例では、"//*" によって XML ツリー全体が session という名前のルール・セッションにアサートされます。

例 4-7 XML 文書のアサート

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

関連項目： 4-25 ページ「JAXB コンテキストの作成と XML 文書のアンマーシャリング」

4.8.8 ルール・セッションでの実行関数の使用

例 4-8 に、ルール・セッションを実行するコードを示します。

例 4-8 Oracle Rules Engine セッションの実行

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

4.9 テスト・プログラムを使用した XML 版レンタカー・サンプルの実行

$HowToDir*/lib ディレクトリには、TestXML.jar が格納されています。これは、CarRentalxml ディクショナリを使用する Oracle Business Rules Java アプリケーションで、ただちに実行できます。ディクショナリ名を変更して、TestXML.java を修正する必要がある場合、ソースは *$HowToDir/src ディレクトリにあります。このディレクトリの Readme.txt ファイルには、テスト・プログラムの実行に必要な環境変数の設定について指示が記載されています。*$HowToDir* には、XML 版 How-To をインストールしたディレクトリが入ります。

例 4-9 に、実行している TestXML からの出力を示します。

例 4-9 レンタカー・プログラムの実行サンプル（XML）

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

すべてのファクトが出力を生成したり、ルールを起動するとはかぎりません。この例では、UnderAge ルールと一致してアサートされたファクトについてのみ出力が表示されます。

5

JSR-94 の使用

この章の内容は次のとおりです。

- Oracle Business Rules での JSR-94 ルール実行セットの使用
- JSR-94 インタフェースを Oracle Business Rules とともに使用

5.1 Oracle Business Rules での JSR-94 ルール実行セットの使用

Rule Author または RL Language テキストのいずれかを使用して作成されたルールとともに JSR-94 を使用するには、それらのルールを JSR-94 ルール実行セットにマップする必要があります。JSR-94 ルール実行セット（ルール実行セット）は、一括して実行することを目的としたルールのコレクションです。ルール実行セットは、実行する前に登録する必要があります。この登録によって、ルール実行セットが URI に関連付けられます。JSR-94 ルール・セッションは、この URI を使用して作成できます。

この項の内容は次のとおりです。

- ファイル・リポジトリのルールセットからの JSR-94 ルール実行セットの作成
- WebDAV リポジトリからの JSR-94 ルール実行セットの作成
- Oracle Business Rules RL Language テキストからのルール実行セットの作成
- URL で指定した RL Language テキストからのルール実行セットの作成
- 複数のソースからのルールセットを持つルール実行セットの作成

注意： Oracle Business Rules では、JSR-94 ルール実行セットの登録は永続的ではありません。したがって、ルール実行セットは、JSR-94 `RuleExecutionSetProvider` インタフェースを使用し、プログラム上で登録する必要があります。

5.1.1 ファイル・リポジトリのルールセットからの JSR-94 ルール実行セットの作成

Rule Author を使用して作成したルールは、ディクショナリ・ストレージ・プラグインを使用してディクショナリに保存できます。Rule Author を使用して作成したルールとともに JSR-94 を使用するには、Rule Author ディクショナリとその内容を JSR-94 ルール実行セットにマップする必要があります。

Rule Author ディクショナリを JSR-94 とともに使用する手順は、次のとおりです。

1. Rule Author ディクショナリのマッピング情報を XML 文書に指定します。表 5-1 に、ルール実行セットの作成に必要なマッピング要素を示します。例 5-1 には、サンプルの XML マッピング・ファイルを示します。
2. 次に、この XML 文書を JSR-94 管理 API で使用して、ルール実行セットを作成します。作成したルール実行セットは、`RuleAdministration` インスタンスを使用して、JSR-94 ランタイムに登録されます。

表 5-1 ファイル・リポジトリの JSR-94 の XML マッピング要素

要素	説明
`<repository-location>`	ファイル・リポジトリ・パス：このパスは、実行時の現行ディレクトリへの絶対パスまたは相対パスにできます。
`<dictionary-name>`	ディクショナリ名。
`<dictionary-version>`	ディクショナリのバージョン。
`<ruleset-list>`	ディクショナリから抽出する Rule Author ルールセットのリスト。ルールセットは相互の依存性を解決するために、解析される順にリストされます。 **注意：**データ・モデルに関連付けられたルールセットは `<ruleset-list>` 要素に含まれません。JSR-94 の実装では、この要素にルールセットがリストされる前に、データ・モデル・ルールセットが Rules Engine にロードされます。
`<ruleset-stack>`	初期ルールセット・スタックを構成するルールセットのリストを指定します。リストに指定するルールセットの順序は、スタックの最上位から最下位です。

例 5-1 ファイル・リポジトリの JSR-94 XML マッピング・ファイル

```
<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>
```

関連項目： *$ORACLE_HOME*/rules/lib/jsr94_obr.jar の XSD ファイル (oracle/rules/jsr94/admin/jsr94-runtime-configuration-1.0.xsd にあります)

5.1.2 WebDAV リポジトリからの JSR-94 ルール実行セットの作成

Rule Author を使用して作成したルールは、ディクショナリ・ストレージ・プラグインを使用して WebDAV リポジトリに保存できます。WebDAV リポジトリに格納されたルールとともに JSR-94 を使用するには、1 つ以上のルールセットを WebDAV リポジトリから JSR-94 ルール実行セットにマップする必要があります。

ファイル・リポジトリに格納されたルールを JSR-94 とともに使用する手順は、次のとおりです。

1. WebDAV リポジトリのマッピング情報を XML 文書に指定します。表 5-2 に、ルール実行セットの作成に必要なマッピング要素を示します。例 5-2 には、サンプルの XML マッピング・ファイルを示します。
2. 次に、この XML 文書を JSR-94 管理 API で使用して、ルール実行セットを作成します。作成したルール実行セットは、RuleAdministration インスタンスを使用して、JSR-94 ランタイムに登録されます。

表 5-2 JSR-94 の WebDAV リポジトリの XML マッピング要素

要素	説明
<repository-url>	WebDAV リポジトリの URL。
<proxy-host>	プロキシ・ホスト名（プロキシがある場合）。これは、オプション要素です。
<proxy-port>	プロキシ・ポート（プロキシがある場合）。これは、オプション要素です。
<dictionary-name>	ディクショナリ名。
<dictionary-version>	ディクショナリのバージョン。

表 5-2 JSR-94 の WebDAV リポジトリの XML マッピング要素（続き）

要素	説明
<ruleset-list>	ディクショナリから抽出する Rule Author ルールセットのリスト。ルールセットは相互の依存性を解決するために、解析される順にリストされます。 **注意 :** データ・モデルに関連付けられたルールセットは <ruleset-list> 要素に含まれません。JSR-94 の実装では、この要素にルールセットがリストされる前に、データ・モデル・ルールセットが Rules Engine にロードされます。
<ruleset-stack>	初期ルールセット・スタックを構成するルールセットのリストを指定します。リストに指定するルールセットの順序は、スタックの最上位から最下位です。

例 5-2 WebDAV リポジトリの JSR-94 マッピング・ファイル

```
<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 Oracle Business Rules RL Language テキストからのルール実行セットの作成

JSR-94 は、テキストとして保存された RL Language ルールセットとともに使用できます。この場合、RL Language テキストはルール実行セットに直接挿入されます。

RL Language で指定したルールを JSR-94 とともに使用する手順は、次のとおりです。

1. RL Language のマッピング情報を XML 文書に指定します。表 5-3 に、ルール実行セットの作成に必要なマッピング要素を示します。例 5-3 には、RL Language テキストを JSR-94 ルール実行セットにマッピングするためのサンプルの XML 文書を示します。
2. 次に、この XML 文書を JSR-94 管理 API で使用して、ルール実行セットを作成します。作成したルール実行セットは、RuleAdministration インスタンスを使用して、JSR-94 ランタイムに登録されます。

表 5-3 JSR-94 の Oracle Business Rules RL Language テキスト XML マッピング要素

要素	説明
<rule-source>	<rl-text> タグが含まれます。このタグには、Oracle Business Rules ルールセットを含む明示的な RL Language テキストが含まれます。複数の <rule-source> タグを使用して、複数のルールセットを指定できます（ルールセットは解析される順に指定します）。
<ruleset-stack>	初期ルールセット・スタックを構成するルールセットのリストを指定します。リストに指定するルールセットの順序は、スタックの最上位から最下位です。

注意： <rl-text> 要素に、事前定義済の XML エンティティを含めることはできません。これには、「&」、「>」、「<」、「"」および「¥」の各文字が含まれます。

例 5-3 RL プログラムで定義されたルールセットの XML マッピング・ファイル

```
<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>
```

関連項目： RL Language テキストを挿入する際に使用できる JSR-94 拡張機能については、5-8 ページの「拡張された createRuleExecutionSet を使用したルール実行セットの作成」を参照してください。

5.1.4 URL で指定した RL Language テキストからのルール実行セットの作成

JSR-94 は、URL を使用して指定した RL Language ルールセットとともに使用できます。

RL Language で指定したルールとともに JSR-94 を使用する手順は、次のとおりです。

1. RL Language のマッピング情報を XML 文書に指定します。表 5-4 に、ルール実行セットの作成に必要なマッピング要素を示します。例 5-4 には、RL Language テキストを JSR-94 ルール実行セットにマッピングするためのサンプルの XML 文書を示します。
2. 次に、この XML 文書を JSR-94 管理 API で使用して、ルール実行セットを作成します。作成したルール実行セットは、`RuleAdministration` インスタンスを使用して、JSR-94 ランタイムに登録されます。

表 5-4 JSR-94 の Oracle Business Rules RL Language URL XML マッピング要素

要素	説明
`<rule-source>`	`<rl-url>` タグが含まれます。このタグには、RL Language テキストの場所を指定する URL が含まれます。複数の `<rule-source>` タグを使用して、複数のルールセットを指定できます（ルールセットは解析される順に指定します）。
`<ruleset-stack>`	初期ルールセット・スタックを構成するルールセットのリストを指定します。リストに指定するルールセットの順序は、スタックの最上位から最下位です。

例 5-4 URL で定義されたルールセットの XML マッピング・ファイル

```
<?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>
```

関連項目： URL を指定する際に使用できる JSR-94 拡張機能については、5-8 ページの「拡張された createRuleExecutionSet を使用したルール実行セットの作成」を参照してください。

5.1.5 複数のソースからのルールセットを持つルール実行セットの作成

ルール実行セットには、複数のソースから導出されたルールを含めることができます。そのソースとして、Rule Author で定義されたルールセットと RL Language ルールセットが混在しても構いません。この場合、XML 要素 `<rule-execution-set>` セットには、複数の `<rule-source>` 要素を含めることができます（ルールの各ソースに対して 1 つ）。各 `<rule-source>` は、Rules Engine で解析される順にリストする必要があります。

注意： このリリースの Oracle Business Rules では、JSR-94 ルール実行セットで参照できるのは 1 つの Rule Author ディクショナリのみです。

5.2 JSR-94 インタフェースを Oracle Business Rules とともに使用

この項では、JSR-94 インタフェースに関して Oracle Business Rules 固有の事項を説明します。この項の内容は次のとおりです。

- CreateRuleExecutionSet を使用したルール実行セットの作成
- createRuleSession を使用したルール・セッションの作成
- JSR-94 メタデータの使用
- Oracle Business Rules の JSR-94 拡張機能の使用

5.2.1 CreateRuleExecutionSet を使用したルール実行セットの作成

`javax.rules.admin` の `RuleExecutionSetProvider` および `LocalRuleExecutionSetProvider` インタフェースには、`RuleExecutionSet` オブジェクトを作成するための `createRuleExecutionSet` が含まれています。

`createRuleExecutionSet` メソッドを使用するために、最初の引数は表 5-5 に示すように解析されます。

表 5-5 createRuleExecutionSet メソッドの最初の引数の種類

引数	説明
`org.w3c.dom.Element`	構成スキーマからの `<rule-execution-set>` 要素のインスタンスを指定します。
`java.lang.String`	構成スキーマからの `<rule-execution-set>` 要素のインスタンスである XML 文書の場所の URL を指定します。
`java.io.InputStream`	構成スキーマからの `<rule-execution-set>` 要素のインスタンスである XML 文書を読み込むために使用する入力スキームを指定します。
`java.io.Reader`	構成スキーマからの `<rule-execution-set>` 要素のインスタンスである XML 文書を読み込むために使用する文字読取装置を指定します。

注意： JSR-94 には、`java.lang.Object` 引数を取る `createRuleExecutionSet` メソッドも含まれます。この引数は、ルール実行セットの抽象的な構文ツリー用です。このリリースの Oracle Business Rules では、この引数を取るメソッドの使用はサポートされていません。`java.lang.Object` 引数を使用してメソッドを起動すると、`RuleExecutionSetCreateException` 例外が発生します。

---

createRuleExecutionSet メソッドの第 2 引数は、ベンダー固有プロパティの java.util.Map です。表 5-6 に、Oracle JSR-94 実装に有効なプロパティを示します。

表 5-6 createRuleExceptionSet の Oracle 固有プロパティ

プロパティ・キー	プロパティ値
oracle.rules.jsr94.sensitiveDataCallback	このプロパティは、選択したリポジトリ（たとえば、認証を要求するように構成された WebDAV サーバー）で認証が必要な場合に設定されます。プロパティ値は、oracle.rules.sdk.repository.SensitiveDataCallback インタフェースの実装である必要があります。

5.2.2 createRuleSession を使用したルール・セッションの作成

クライアントでは、RuleRuntime クラスの createRuleSession メソッドを使用して、JSR-94 ルール・セッションを作成します。このメソッドは、ベンダー固有プロパティの java.util.Map 引数を取ります。この引数は、Oracle Business Rules oracle.rules.rl.RuleSession に対して定義されたプロパティを渡すために使用できます。ルール実行セットに URL またはリポジトリ・ルール・ソースが含まれる場合、そのソースからのルールは、新規の RuleSession が作成されるたびに取得されます。

5.2.3 JSR-94 メタデータの使用

JSR-94 では、ルール実行セットおよびルール実行セット内のルールに対して、メタデータを使用できます。Oracle Business Rules 実装では、JSR-94 仕様以外のメタデータは追加されません。ルール実行セットの説明はオプション項目であるため、説明がない場合があります。説明がない場合は、空の文字列が戻されます。ルールについては、ルール名のみが使用可能で、説明は空の文字列で初期化されます。

5.2.4 Oracle Business Rules の JSR-94 拡張機能の使用

この項では、JSR-94 実装クラスで提供される次の拡張機能について説明します。

- 拡張された createRuleExecutionSet を使用したルール実行セットの作成
- JSR-94 での RL Language 関数の起動

5.2.4.1 拡張された createRuleExecutionSet を使用したルール実行セットの作成

Oracle Business Rules には、RuleExecutionSet を作成する際に入力として必要な XML 制御ファイルを簡単に作成するために、ヘルパー関数が用意されています。

RLLocalRuleExecutionSetProvider クラスのヘルパー・メソッド createRuleExecutionSet を使用できます。createRuleExecutionSet メソッドには、次のシグネチャがあります。

```
RuleExecutionSet createRuleExecutionSet(String name,
                                        String description,
                                        RuleSource[] sources,
                                        String[] rulesetStack,
                                        Map properties)
```

表 5-7 に、createRuleExecutionSet の引数を示します。

表 5-7 createRuleExecutionSet の引数

引数	説明
name	ルール実行セットの名前を指定します。
description	ルール実行セットの説明を指定します。
sources	ルールのソースを指定する配列を指定します。このリリースでは、RL Language テキスト、RL Language テキストへの URL、ファイル・リポジトリ（.jar ファイル）および WebDAV リポジトリの 4 種類のソースがサポートされています。RuleSource は、RLTextSource（RL Language テキスト）、RLUrlSource（RL Language URL）、JarRepositorySource（ファイル・リポジトリ）および WebDAVRepositorySource（WebDAV リポジトリ）の各クラスによって実装されるインタフェースです。 詳細は、『Oracle Business Rules Java API Reference』の oracle.rules.jsr94.admin パッケージに関する説明を参照してください。
rulesetstack	ルールの実行前に設定される RL Language ルールセット・スタックの初期内容を指定します。配列の内容は、スタックの最上位（0 番目の要素）から最下位（最後の要素）の順に指定します。
properties	Oracle 固有プロパティ。表 5-6 を参照してください。

5.2.4.2 JSR-94 での RL Language 関数の起動

JSR-94 ルール・セッションとのステートフルな相互作用では、ユーザーが任意に RL Language 関数を起動できることが必要です。JSR-94 StatefulRuleSession インタフェースを実装するクラスによって、oracle.rules.rl.RuleSession クラスの callFunction メソッドにアクセスできます。

例 5-5 に、JSR-94 StatefulRuleSession で引数なしの RL Language 関数を起動する方法を示します。

例 5-5 StatefulRuleSession での CallFunction の使用

```
import javax.rules.*;
 ...
StatefulRuleSession session;
...
((oracle.rules.jsr94.RLStatefulRuleSession) session).callFunction("myFunction");
```

6

Oracle Business Rules SDK の使用

Oracle Business Rules SDK (Rules SDK）には、開発者が、カスタマイズ・アプリケーションの記述に使用できる API が用意されています。このアプリケーションは、ルールとデータ・モデル（および Oracle Business Rules ディクショナリに格納されているすべての情報）へのアクセス、作成または変更に使用します。Rules SDK API を使用すると、適切に定義されたインタフェースを使用して、ディクショナリ・データを作成、変更およびアクセスできます。この API は、カスタマイズ・ルール対応アプリケーションの構築にも使用できます。

Rules SDK API は、既存のルールにアクセスし、Rules Engine を実行するルール対応アプリケーションで使用したり、ルールとデータ・モデルの情報へのアクセス、作成または編集を記述したアプリケーションで使用できます。

この章では、Oracle Business Rules SDK API について説明します。

この章の内容は次のとおりです。

- Rules SDK の構築ブロック
- リポジトリとディクショナリの使用
- データ・モデルの使用
- ルールセットの使用およびルールの作成と変更

6.1 Rules SDK の構築ブロック

最上位レベルの Rules SDK パッケージである oracle.rules.sdk には、次の各パッケージが含まれています。

- oracle.rules.sdk.repository
- oracle.rules.sdk.dictionary
- oracle.rules.sdk.editor.datamodel
- oracle.rules.sdk.editor.ruleset
- oracle.rules.sdk.exception

JavaBean モデルに準拠した Rules SDK インタフェースには、Bean プロパティごとに getter と setter が組み込まれています。たとえば、setName("somevalue") によって、個別インスタンスの name プロパティが設定されます。

Bean インタフェースに加えて、Rules SDK には、ハッシュ関数の get および put スタイル・インタフェースが用意されています。一般的に、Bean インタフェースは有用ですが、少なくとも 1 つの GUI フレームワークで HashMap スタイルが必要になります。

6.2 リポジトリとディクショナリの使用

Oracle Business Rules ディクショナリは、リポジトリに格納されています。ディクショナリにアクセスする前に、そのリポジトリへのアクセスを確立する必要があります。このためには、アクセスするリポジトリのタイプと、そのリポジトリ固有のタイプに必要な初期化パラメータを指定する必要があります。出荷時の Rule Author では、WebDAV（Web Distributed Authoring and Versioning）リポジトリとファイル・リポジトリがサポートされています。

表 6-1 に、WebDAV リポジトリの初期化パラメータ・キーを示します。リポジトリ・タイプ・キーは、oracle.rules.sdk.store.webdav です。

表 6-1 WebDAV リポジトリ・タイプの初期化パラメータ・キー

パラメータ	キー	説明
URL	oracle.rules.sdk.store.webdav.url	希望する WebDAV ルール・リポジトリの URL。このパラメータは必須です。
プロキシ・ホスト	oracle.rules.sdk.store.webdav.proxyHost	プロキシ・サーバーのホスト名。Rule Author が実行されているサーバーと WebDAV サーバーとの間にプロキシ・サーバーがある場合のみ必須です。
プロキシ・ポート	oracle.rules.sdk.store.webdav.proxyPort	プロキシ・サーバーに使用するポート。Rule Author が実行されているサーバーと WebDAV サーバーとの間にプロキシ・サーバーがある場合のみ必須です。

表 6-2 に、ファイル・リポジトリの初期化パラメータ・キーを示します。リポジトリ・タイプ・キーは、oracle.rules.sdk.store.jar です。

表 6-2 ファイル・リポジトリ・タイプの初期化パラメータ・キー

パラメータ	キー	説明
ファイル・パス	oracle.rules.sdk.store.webdav.path	ルール・リポジトリが保存されているファイルへのパス。このパラメータは必須です。

6.2.1 WebDAV リポジトリへの接続の確立

例 6-1 に、WebDAV リポジトリへのアクセスの確立方法を示します。

例 6-1 WebDAV リポジトリへのアクセスの確立

```
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);
```

WebDAV リポジトリが認証を必要とするように構成されている場合は、次の手順を実行する必要があります。

- 必要なユーザー名とパスワードを使用して、ウォレットを構成します。
- `oracle.rules.sdk.callbacks.WalletCallback` クラスのインスタンスを作成し、`init` メソッドをコールする前に、`RepositoryContext` 内に設定します。

例 6-2 の `/wallets/rules_wallet` は、WebDAV 認証用の資格証明で構成したウォレットへのパスです。

例 6-2 認証用のウォレットの構成

```
WalletCallback callback = new WalletCallback("/wallets/rules_wallet", null);
rc.setSensitiveDataCallback(callback);
```

6.2.2 ファイル・リポジトリへの接続の確立

例 6-3 に、ファイル・リポジトリへのアクセスの確立方法を示します。

例 6-3 ファイル・リポジトリへのアクセスの確立

```
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 ディクショナリのロード

ディクショナリは、ディクショナリ名を指定することでロードできます。この場合は、次のコードを使用して、ディクショナリのデフォルト・バージョンをロードします。

```
RuleDictionary dictionary = repos.loadDictionary(dictionaryName);
```

または、次のコードを使用して、ディクショナリ名とバージョンの両方を指定します。

```
RuleDictionary dictionary = repos.loadDictionary(dictionaryName,
                                                 dictionaryVersion);
```

これらの例は、ファイル・リポジトリと WebDAV リポジトリの両方に適用できます。

6.3 データ・モデルの使用

Rules SDK データ・モデルには、ルールの作成に使用するファクト・タイプ、内部変数、制約および関数が含まれています。データ・モデルのファクト・タイプは、対応するルールに基づいて判断できます。Oracle Business Rules の変数には、複数のルールで共有する情報が含まれています。Oracle Business Rules の関数には、ルールに対して再使用するロジックが用意されています。制約によって、ルールのカスタマイズに使用する有効な値のセットが制限されます。

注意： 既存の Java クラスまたは XML スキーマをインポートするには、Rule Author アプリケーションを使用する必要があります。SDK でリポジトリを使用できるのは、Rule Author を使用してクラスまたはスキーマをインポートした後です。SDK の今後のバージョンでは、Java クラスおよび XML スキーマを直接インポートできるように機能が拡張される予定です。

リポジトリを使用して `RuleDictionary` オブジェクトを作成または開いた後は、Rules SDK を使用してこのディクショナリにデータ・モデルを作成できます。`RuleDictionary` オブジェクトは、`DataModel` インスタンスの作成に必要な内部データ構造にアクセスできます。

この章で紹介するデータ・モデルの例には、`email` というサンプル・パッケージからインポートした Java ファクト・タイプが含まれています。この `email` パッケージは、Rule Author を使用してインポートしました。表 6-3 に示したクラスとプロパティは、`email.jar` をインポートして移入しました。

表 6-3 サンプルの `email` パッケージ・クラス

名前	説明	タイプ
`email.ElectronicMessage`	メッセージの発生を表します。	Java ファクト・タイプ
`email.EmailAddress`	電子メール・アドレスを表します。	Java ファクト・タイプ
`email.EmailAddressList`	電子メール・アドレスのリストを表します。	Java ファクト・タイプ

スパムを処理するデータ・モデル例では、`email` パッケージが使用され、データ・モデルは、`SpamFound` という `RLFact` オブジェクトを作成することで推論をサポートします。グローバル・カウントを組み込むには、`spamCounter` という変数を作成し、定数の `String` 変数によってスパムを示します。ルールによって、スパムの電子メール・メッセージが検出されると、`killSpam` という関数からアクションが提供されます。表 6-4 に、これらのデータ・モデル・コンポーネントを示します。

表 6-4 email パッケージを処理するためのサンプル・データ・モデル・タイプ

名前	説明	タイプ
`SpamFound`	電子メール・メッセージがスパムであると判断された場合にアサートされます。	RL ファクト・タイプ
`spamCounter`	スパム・メッセージの件数を累計します。	変数
`SpecialOffer_CONST`	特別なオファーの `String` を含む定数です。	定数変数
`fKillSpam`	削除するスパムが検出されるとコールされます。	RL 関数

6.3.1 データ・モデルの作成

RuleDictionary オブジェクトを作成して開いた後は、Rules SDK を使用して editor.DataModel インスタンスを作成できます。RuleDictionary オブジェクトは、DataModel インスタンスの作成に必要な内部データ構造にアクセスできます。

次に例を示します。

```
eDM = new oracle.rules.sdk.editor.datamodel.DataModel(m_dict);
```

データ・モデル・タイプ・システムの基本は FactType オブジェクトです。FactType オブジェクトは、プリミティブ、Java、XML または RL ファクト・タイプとして定義されます。プリミティブ・ファクト・タイプは固定で、Java プリミティブ（String、int、double など）が含まれます。Rules SDK では、RuleDictionary を作成すると、プリミティブ・タイプが自動的に作成されます。JAR ファイル、クラス・ファイルまたはスキーマ・ファイルからクラスをインポートするときは、Java および XML のファクト・タイプを作成します。RL ファクト・タイプは、Rules SDK を使用して直接作成できます。Java、XML および RL の各ファクト・タイプではクラスが定義され、JavaBean で定義されたプロパティを表す関連のプロパティとメソッドを指定できます。

6.3.2 データ・モデル・コンポーネントの作成

データ・モデルの各部分は、適切な ModelComponentTable オブジェクトを使用して作成します。新規インスタンス（Function、FactType、Variable または Constraint）の作成に必要な順序は、次のとおりです。

- データ・モデル内の適切な表をインスタンス化します。
- 表のインスタンスの add() メソッドを起動します。

例 6-4 に、関数インスタンスの作成方法を示します。

例 6-4 関数インスタンスの作成

```
FunctionTable ft = eDM.getFunctionTable();
Function fKillSpam = ft.add();
```

既存の Java クラスまたは XML スキーマをインポートするには、Rule Author アプリケーションを使用する必要があります。SDK でリポジトリを使用できるのは、Rule Author を使用してクラスまたはスキーマをインポートした後です。SDK の今後のバージョンでは、Java クラスおよび XML スキーマを直接インポートできるように機能が拡張される予定です。

6.3.3 関数の引数リストの作成

Rules SDK では、FormalParameterTable オブジェクトを使用して、関数の引数リストを作成します。各 FormalParameter エントリは、1 つのパラメータを表します。1 番目のパラメータは、FormalParameterTable 内の 1 番目（ゼロ）のエントリ、2 番目のパラメータは 2 番目のエントリと、以下同様に続きます。変数が定数の場合があります。この場合は、最初の初期化後に、値を割り当てることはできません。setFinal() メソッドは、定数の動作を制御します。各 FormalParameter オブジェクトには、タイプと名前があります。仮パラメータのタイプは、使用可能なファクト・タイプから選択されます。

例 6-5 のコードでは、email サンプルに対する FormalParameterTable オブジェクトが作成されます。

例 6-5 FormalParameterTable の作成

```
// 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 初期化式の作成

Rules SDK では、変数にルールセットの内部の状態が組み込まれます。各変数には、タイプと初期化式を指定する必要があります。変数タイプは、使用可能なファクト・タイプ（データ・モデルにはすべてのファクト・タイプが定義されています）のリストから選択します。

式には、次の 2 つのタイプがあります。

- `Expression`

  このタイプの式は、次の形式に従う必要があります。

  operand operator operand

  `Expression` で使用可能なオペランドと演算子は、`AdvancedExpression` で使用可能なオペランドと演算子よりも制限されています。たとえば、`Expression` では、オペランドに、パラメータが必要な関数を使用できません。

- `AdvancedExpression`

  このタイプの式では、すべてのオペランドと演算子が使用できます。

式の初期化には、`AdvancedExpression` を使用することをお薦めします。例 6-6 に、変数の初期値の設定方法を示します。

例 6-6 変数の初期値の設定

```
InitialValue iv = var.getValue();
AdvancedExpression adv = iv.getAdvancedExpression();
adv.insert(0, "¥"FIXEDVALUE¥"");
```

6.3.5 RL 関数本体の作成

RL 関数本体は、RL Language の複数の文字列で構成されます。RL 関数本体は、任意の関数パラメータ、つまり任意のグローバル変数を参照する場合があります。関数本体は文字列で入力します。この文字列は、構文として正しい RL Language である必要があります（例 6-7 を参照）。

例 6-7 関数本体の作成

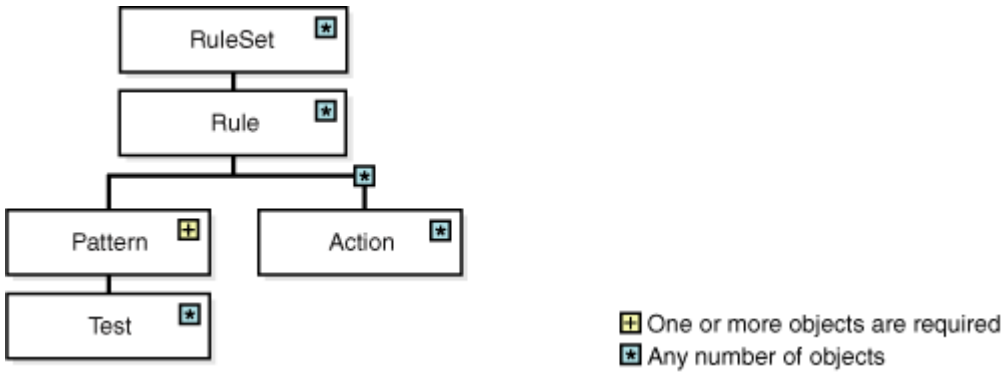
```
//set the body of the function
// in this case just pretty print a message
fKillSpam.setBody(" println(¥" email from: ¥" + emsg.getSender() + ¥"  because ¥" +
reason)" );
```

6.4 ルールセットの使用およびルールの作成と変更

Rules SDK では、ルールは（**条件**と呼ばれる）条件式で、条件が true と評価された場合に実行される一連のアクションです。ルール条件は、一連のパターンで構成されます。パターンによって一致タイプが決まり、そのタイプと、前のパターン（順序に意味があります）に出現した他のタイプに対するテストが組み込まれます。ルール・アクションには、コール、割当て、取消しおよびアサーションがあります。

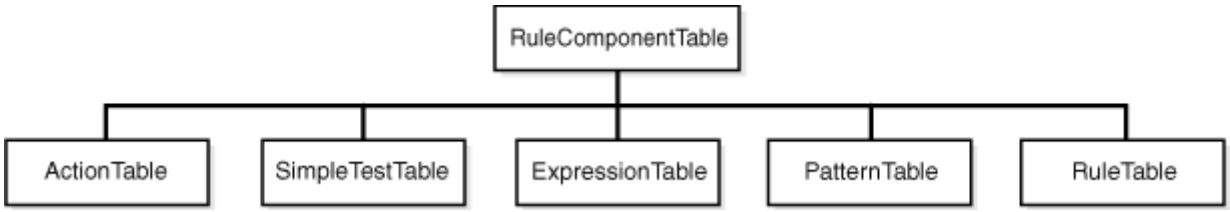
図 6-1 に、ルールセットの一般的なコンテナ階層を示します。

図 6-1 Rules SDK ルールセットのコンテナ階層



Rules SDK には、これらの各オブジェクトを表すクラスが用意されています。すべてのクラスは、`RuleComponent` の下位クラスです。コレクション（ルールセット内のルール、ルール内のパターン、ルール内のアクションなど）の場合は、特定のコレクションを管理するために、Rules SDK によって、`RuleComponentTable` より下位のクラスが提供されます（図 6-2 を参照）。`RuleComponentTable` サブクラスでは、表で表す特定のサブクラスに対して特別な `add()` メソッドが提供されます。

図 6-2 Rules SDK の RuleComponent



XML スキーマ、Java クラス、RL グローバル変数および RL 関数を記述した Rules SDK コンポーネントは、（ディクショナリに格納された）データ・モデル内にあります。通常、`RuleComponent` インスタンスは、データ・モデル・エンティティを参照します。

6.4.1 ルールセットの作成

一般的に、RuleComponent オブジェクトの作成に使用する順序は、次のとおりです。

1. RuleSet コンストラクタを使用して、RuleSet インスタンスを作成します。
2. 適切な表（PatternTable、ActionTable、ExpressionTable、SimpleTestTable）の add() メソッドによって、RuleSet の子を作成します。

Rules SDK API には、RuleSet 内のコレクションを表すクラスが用意されています。これらの各オブジェクトは、RuleComponent の下位オブジェクトです。コレクション（ルールセット内のルール、ルール内のパターンまたはルール内のアクションなど）の場合は、特定のコレクションを管理するために、Rules SDK によって、RuleComponentTable クラスより下位のクラスが提供されます。RuleComponentTable サブクラスでは、表で表す特定のサブクラスに対して特別な add() メソッドが提供されます。

例 6-8 のコードでは、email サンプルに対する RuleSet が作成されます。

例 6-8 ルールセットの作成

```
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 ルールセットへのルールの追加

Rules SDK には、RuleSet 内のオブジェクトを表すクラスが用意されています。これらの各オブジェクトは、RuleComponent オブジェクトの下位オブジェクトです。コレクション（ルールセット内のルール、ルール内のパターンまたはルール内のアクションなど）の場合は、特定のコレクションを管理するために、Rules SDK によって、RuleComponentTable クラスより下位のクラスが提供されます。RuleComponentTable サブクラスでは、表で表す特定のサブクラスに対して特別な add() メソッドが提供されます。

コレクションの一部ではないクラスがいくつかあります。これらのクラスにアクセスするには、getter インタフェースで親 Bean を使用します。たとえば、適切な Pattern インスタンスで getAdvancedExpression() を起動し、AdvancedExpression を取得します。

例 6-9 のコードは、ルールセット内の表にルールを追加する方法を示しています。

例 6-9 ルールセット内の表へのルールの追加

```
//add rule to the table
oracle.rules.sdk.editor.ruleset.Rule r = rs.getRuleTable().add();
r.setName("DetectSpamRule");
```

6.4.3 ルールへのパターンの追加

Rules SDK API には、パターンを表す Pattern クラスが用意されています。getPatternTable サブクラスには、例 6-10 に示すように、パターン・オブジェクトを追加するための特別な add() メソッドが用意されています。

例 6-10 ルールへのパターンの追加

```
//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");
```

TestForm プロパティでは、パターンに関連付けられたテストのタイプが定義されます。

6.4.4 パターンへのテストの追加

すべての Pattern には、その Pattern に関連したテストを指定できます。テストには、SimpleTest オブジェクトまたは AdvancedExpression オブジェクトの形式を採用できます。Pattern に指定できるテスト数に制限はありません。

RL Language の生成時には、Rules SDK の論理積（AND）もテストします。たとえば、email の例に使用するテストには、パラメータ付きの関数が必要です。このためには、AdvancedExpression オブジェクトが必要です（例 6-11 を参照）。

例 6-11 パターンへのテストの追加

```
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
```

テストが次のような形式の場合は、

operand operator operand

どちらのオペランドもパラメータを必要としない関数であるため、SimpleTest を使用できます。たとえば、パターン変数名が emsg で、テストが emsg.sender == david@fun.com の場合、簡易テストでは、例 6-12 のようになります。

例 6-12 簡易テスト

```
// 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 ルールへのアクションの追加

Rules SDK では、次のタイプの `Actions` をサポートしています。

- Assert New
- Assert
- Assign
- Call
- Retract
- RL

アクションの `setForm` プロパティ内の設定によって、アクションのタイプが定義されます。特定のルールに関連付けられた `getActionTable` インスタンスの `add()` メソッドを使用して、アクションを追加します（例 6-13 を参照）。

例 6-13 アクション・タイプの定義

```
//////////////////////////////////////////////////////////////////////
// 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 ルールセットおよびルールの追加に関する注意事項

RuleSet にコンポーネントを追加する順序は重要です。必要な子オブジェクトは、親オブジェクトを使用して作成します。たとえば、RuleSet の RuleTable インスタンスにルールを追加できるのは、RuleSet インスタンスを作成した後です。RuleSet を作成した後で、RuleTable の add() メソッドを使用して、RuleSet にルールを追加します。次に、PatternTable.add() メソッドを使用して、ルールにルール・パターンを追加します。最後に、Pattern 内にテストを作成するには、getAdvancedExpression() を使用して、AdvancedExpression にアクセスするか、標準モードのテストの場合は、SimpleTestTable.add() を使用します。

A

Oracle Business Rules のファイルと制限事項

この付録では、Rule Author のファイルと名前に関する既知のネーミング制約と、Rules SDK の特定の制限事項について説明します。

この付録の内容は次のとおりです。

- Rule Author のネーミング規則
- Rule Author のセッション・タイムアウト
- Rules SDK と Rule Author のテンポラリ・ファイル

A.1 Rule Author のネーミング規則

この項では、Rule Author のネーミング規則について説明します。

A.1.1 ルールセットのネーミング

Rule Author には、ルールセット名に関する制限事項があります。ルールセット名に使用できるのは、文字（a ～ z および A ～ Z）と数字（0 ～ 9）、またはアンダースコア文字（_）のみです。

A.1.2 ディクショナリのネーミング

Rule Author のディクショナリ名に使用できるのは、大文字と小文字（a ～ z および A ～ Z）、数字（0 ～ 9）、ピリオド（.）、アンダースコア文字（_）およびハイフン（-）のみです。特殊文字は使用できません。

Rule Author のディクショナリ名では、大 / 小文字は保持されますが、区別されません。これは、Dictionary と DICT の両方のディクショナリ名が有効であることを意味します。また、Test という名前のディクショナリを作成している場合、TEST という名前の別のディクショナリは、最初に Test というディクショナリを削除した場合のみ作成できます。

また、ディクショナリ名には、1 つ以上の文字を含める必要があります。たとえば、1.1 という名前のディクショナリは有効ではありませんが、Version1.1 は有効です。

A.1.3 バージョンのネーミング

Rule Author には、バージョン名に関する制限事項があります。バージョン名に使用できるのは、文字（a ～ z および A ～ Z）と、数字（0 ～ 9）、またはアンダースコア文字（_）のみです。特殊文字は使用できません。

Rule Author のバージョン名では、大 / 小文字は保持されますが、区別されません。これは、Version と VERS の両方のバージョン名が有効であることを意味します。また、Test という名前のバージョンを作成している場合、TEST という名前の別のバージョンは、最初に Test というバージョンを削除した場合のみ作成できます。

A.1.4 別名のネーミング

Rule Author の別名には、単一のスペースも含めてすべての文字を使用できます。式で使用する別名の先頭が文字、ドル記号（$）またはアンダースコア（_）で、次にこれらの文字のみが使用されている場合は、別名を引用符で囲む必要はありません。

特殊文字や埋め込まれたスペースを含む別名を拡張式で使用する場合、その別名は、逆引用符（'）文字で囲む必要があります。たとえば、別名の Driver@ は、次のように指定する必要があります。

```
'Driver@'
```

A.1.5 XML スキーマのターゲット・パッケージのネーミング

「XML スキーマ・セレクタ」ページ上で、XML ファクトに指定する**ターゲット・パッケージ名**は、ASCII 文字、数字およびアンダースコア文字に制限されます。

A.2 Rule Author のセッション・タイムアウト

非アクティブ期間後は Rule Author セッションがタイムアウトになるため、作業に応じて定期的にディクショナリを保存してください。タイムアウト期間は、`<session-timeout>` 要素を使用して Rule Author アプリケーションの `web.xml` ファイルに指定します。

Rule Author がタイムアウトになると、ロードされたディクショナリの現在の作業がリポジトリ内の `SCRATCH_<login user name>` という名前のバージョンに自動的に保存されます（*login user name* は、Rule Author にログインしているユーザーの名前です）。

Rule Author はユーザーごとに `SCRATCH_` ファイルを 1 つしか保存しません。したがって、Rule Author が一度タイムアウトになった場合、`SCTATCH_` バージョンはログイン・ユーザー名を使用してリポジトリに保存されます。同じユーザーが再度ログインし、同じリポジトリに接続すると、Rule Author によって、`SCRATCH` バージョンを新しいバージョン名に保存する必要があることを示す警告メッセージが表示されます。Rule Author が再度タイムアウトになった場合で、`SCRATCH` バージョンを別のバージョンに保存しない場合、2 回目以降のタイムアウトによって、既存の `SCRATCH` バージョンが（Rule Author ユーザーごとに）上書きされます。

Rule Author がタイムアウトになると、Oracle Single Sign-On ログイン・ページにリダイレクトされます。Oracle Single Sign-On が使用可能でない場合、タイムアウト後に Rule Author のデフォルト認証ページにリダイレクトされます。

関連項目： C.9 項「Oracle Application Server での Single Sign-On タイムアウトを変更するにはどうすればよいですか。」

A.3 Rules SDK と Rule Author のテンポラリ・ファイル

ファイル・リポジトリを更新すると、Rules SDK によってテンポラリ・ファイルが作成および使用されます。通常の動作条件では、テンポラリ・ファイルを使用する操作が完了すると、Rules SDK によって、これらのテンポラリ・ファイルがシステムから削除されます。ただし、異常終了の特定の条件によっては、これらのテンポラリ・ファイルがシステムに残る場合があります。

ファイル・リポジトリとテンポラリ・ファイルの詳細は、B.3 項「ファイル・リポジトリの使用」を参照してください。

B

リポジトリを利用した Rule Author と Rules SDK の使用

この付録では、リポジトリを利用した Rule Author と Rules SDK の使用について説明します。

この付録の内容は次のとおりです。

- WebDAV リポジトリの使用
- WebDAV リポジトリのセキュリティ
- ファイル・リポジトリの使用
- リポジトリの高可用性

B.1 WebDAV リポジトリの使用

この項では、WebDAV リポジトリの設定と構成について説明します。

B.1.1 WebDAV リポジトリの設定

Oracle Business Rules では、ルールセット、データ・モデルおよびルールの永続的なストレージとして、WebDAV リポジトリの使用がサポートされています。この項では、WebDAV リポジトリの設定方法を説明し、Oracle HTTP Server に、ファイル・システム・ベースの WebDAV リポジトリを設定するための基本的な手順を説明します。Oracle HTTP Server では、`mod_oradav` モジュールによって WebDAV がサポートされます。

WebDAV プロトコルは HTTP プロトコルの拡張機能で、リモート・ユーザーはこれによって、Web サーバーにコンテンツを書き込むことができます。この構成を使用して、望ましくない結果を防止し、セキュアなシステムを保守するために、Web サーバーを適切に構成することが重要です。

Web サーバーに次のセキュリティ機能の一部またはすべてを採用することをお薦めします。

- WebDAV 対応領域へのアクセスに対して認証を要求します。
- 認証中は少なくとも SSL を使用します（Basic 認証を使用する場合はセッション全体に対して使用します）。
- `ForceType` ディレクティブを使用して、WebDAV 対応領域内のコンテンツを参照する URL の実行を防止します。

次の例は、ファイル・システムにコンテンツが格納されている WebDAV ベースのリポジトリを確立するために使用できる手順を示しています。この例のすべてのファイル・システム・パスは、Oracle HTTP Server がインストールされている *`ORACLE_HOME`* に対する相対パスです。また、この例では、ユーザーが Oracle Application Server をインストールしたユーザーでログインし、Oracle HTTP Server には、URL の *`http://www.myserver.com:port`* を使用してアクセスできることを前提としています。

注意： この例の WebDAV リポジトリの構成は、内部テストとしてのみ使用し、実際の本番環境では使用しないでください。この構成には、アクセス制御が含まれていないため、WebDAV リポジトリに対するアクセスまたは変更が誰に対しても許可されます。WebDAV リポジトリのセキュリティ構成の詳細は、B.2 項を参照してください。

---

1. `Apache/Apache/htdocs` ディレクトリ（フォルダ）にナビゲートします。
2. `rule_repository` という名前のディレクトリを作成します。
3. Oracle HTTP Server による `rule_repository` ディレクトリの読取りと書込みが可能であることを確認します。
4. `Apache/oradav/conf` ディレクトリにナビゲートします。
5. `moddav.conf` ファイルを編集し、次の行を追加します。

```
<Location /rule_repository>
    DAV on
    ForceType text/plain
</Location>
```

6. Oracle HTTP Server を再起動します。

この手順によって、次の URL でアクセス可能な WebDAV リポジトリが確立します。

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

注意： 認証が実行されるように、URL には完全修飾ホスト名を使用する必要があります。

関連項目： mod_oradav の構成と使用については、『Oracle HTTP Server 管理者ガイド』を参照してください。特に、第 9 章の WebDAV に関するセキュリティ上の考慮事項に関する項を参照してください。

B.1.2 WebDAV リポジトリへの接続

WebDAV リポジトリ・タイプを選択すると、Rule Author によって、表 B-1 に示した構成パラメータが表示されます。

表 B-1 WebDAV リポジトリへの接続に関するパラメータの構成

パラメータ	説明
URL	希望する WebDAV ルール・リポジトリの URL。このパラメータは必須です。ホスト名は、完全修飾ホスト名である必要があります。
User Name	WebDAV へのアクセス権限を持つユーザーを指定します。
Password	指定したユーザー名に関連付けられている WebDAV ユーザーのパスワードを指定します。

注意： Rule Author では、ユーザー名、パスワードおよびその他の必須プロパティを指定し、Oracle ウォレットも指定する場合、ダイアログで指定したプロパティが Oracle ウォレット情報に優先します。

B.1.3 プロキシを使用した WebDAV リポジトリへの接続

Rule Author は、http.proxyHost システム・プロパティを検索します。このプロパティが設定されている場合、Rule Author は http プロキシ・システム・プロパティを取得し、WebDAV 接続に使用します。http プロトコル・ハンドラがプロキシを使用するように指定するために設定できるプロパティには、次の 3 つがあります。

- http.proxyHost: プロキシ・サーバーのホスト名。
- http.proxyPort: ポート番号。デフォルト値は 80 です。
- http.nonProxyHosts: プロキシをバイパスして、直接到達するホストのリスト。これは、「|」で区切られた正規表現のリストです。これらの正規表現のいずれかに一致するホストには、プロキシ経由ではなく、直接接続経由で到達します。

WebDAV リポジトリにアクセスするためにプロキシが必要な場合、Rule Author によって、表 B-1 に示すパラメータに加えて、表 B-2 に示すパラメータが表示されます。

表 B-2 プロキシによる WebDAV リポジトリへの接続に関するパラメータの構成

パラメータ	説明
Proxy User Name	プロキシ・ユーザー名を指定します。プロキシ・サーバーにセキュリティが構成されている場合に必須です。
Proxy Password	プロキシ・パスワードを指定します。プロキシ・サーバーにセキュリティが構成されている場合に必須です。

B.2 WebDAV リポジトリのセキュリティ

WebDAV では、WebDAV 対応 Web サーバーに対する読取りと書込みのアクセスが可能です。WebDAV Web サーバーを保護するための適切な手順を実行することをお薦めします。この保護のために、SSL を利用して WebDAV Web サーバーへの接続を暗号化する必要があります。また、認証を要求する必要があります。

内容は次のとおりです。

- Rule Author の SSL を経由した WebDAV リポジトリとの通信
- Oracle ウォレットの場所の設定
- WebDAV リポジトリ認証に対する Rule Author の構成
- Oracle ウォレットへの WebDAV リポジトリ認証に関するデータの格納

B.2.1 Rule Author の SSL を経由した WebDAV リポジトリとの通信

Rule Author が Oracle Application Server 環境にデプロイされている場合、WebDAV リポジトリに対する基本の SSL 接続は Rule Author でサポートされます。必要なことは、入力した WebDAV URL によって `https` を指定することです。

Rule Author が、スタンドアロンの OC4J 環境や HTTP のみをサポートする Oracle 以外のコンテナにデプロイされている場合、WebDAV リポジトリに対する SSL 接続はサポートされません。

Oracle Application Server には、自己署名の SSL 証明書のテストが用意されています。テスト証明書の本番環境での使用は安全でないため、この証明書はユーザー独自の証明書に置換する必要があります。信頼できる認証局の証明書を使用する場合は、OC4J コンテナの内側と外側の両方から WebDAV にアクセスできます。独自の自己署名証明書の使用を選択する場合、コンテナの内側からはアクセスできますが、外側からはアクセスできません。アクセスするためには、デフォルトの JSSE トラスト・ストアを変更する必要があります。詳細は、JDK に含まれている『JSSE Reference Guide』を参照してください。

また、Oracle SSL 実装は、J2SE アプリケーションのクラスパスには提示しないでください。

B.2.2 Oracle ウォレットの場所の設定

Rule Author に対する Oracle ウォレットの場所をカスタマイズする手順は、次のとおりです。

1. Enterprise Manager にログインし、OC4J ホーム・ページに進みます。
2. **「アプリケーション」**タブをクリックします。
3. Rule Author アプリケーションへのリンク（このリンクの名前は、Rule Author アプリケーションを初めてデプロイしたときに定義されています）をクリックします。
4. 「モジュール」表内の **ruleauthor** リンクをクリックします。
5. **「管理」**タブをクリックします。
6. 「マッピング」タスクで、「環境エントリ・マッピング」というラベルの行を検索し、「タスクに移動」列の対応するアイコンをクリックします。
7. `walletStorePath` エントリの「デプロイ済の値」列にウォレットの場所を指定します。
8. Rule Author を再起動します。

ウォレットの場所は、「デプロイ・プランの編集」をクリックし、左側にあるナビゲーション・ツリーを env-entry が表示されるまで開いて、Rule Author をデプロイするときにも設定できます。env-entry を開き、`walletStorePath` を選択します。ウォレットの場所を指定した後は、Rule Author を再起動してください。

B.2.3 WebDAV リポジトリ認証に対する Rule Author の構成

認証を要求するように構成された WebDAV リポジトリに接続しようとする Rule Author は、認証リクエストに応答できる必要があります。リポジトリ認証に対して Rule Author を構成する手順は、次のとおりです。

1. WebDAV リポジトリの適切なユーザー名とパスワードを Oracle ウォレットに格納します。
2. プロキシ・サーバーにも認証が必要な場合は、そのプロキシ・サーバーのユーザー名とパスワードを Oracle ウォレットに格納します。
3. Rule Author 環境エントリを構成して、Oracle ウォレットを指定します（B.2.2 項「Oracle ウォレットの場所の設定」を参照）。
4. Rule Author アプリケーションを再起動します。

B.2.4 Oracle ウォレットへの WebDAV リポジトリ認証に関するデータの格納

WebDAV リポジトリから受信した認証リクエストによって、次の情報が提供されます。

- 認証をリクエストしているサーバーのホスト名
- サーバーのポート
- レルム（Oracle HTTP Server 構成内の `AuthName`）
- プロキシ・サーバーの認証であるかどうかの表示

この情報は、認証用のユーザー名とパスワードを取得するキーの構築に使用されます。プロキシ・サーバーで認証が必要な場合は、複数の認証リクエスト（プロキシ・サーバーと WebDAV サーバーの各リクエスト）を処理できます。

リクエストがプロキシ認証用である場合、そのキーは「proxy-」から始まります。キーの後には、「-」で区切られたホスト名、ポートおよびレルムの各フィールドが順に続きます。最後に、ユーザー名には「-u」が、パスワードには「-p」がキーに追加されます。たとえば、次の内容を想定します。

- ホスト : `myserver.myco.com`
- ポート : 443
- レルム : Authorized WebDAV Users Only
- プロキシ・サーバーの指定 : `wwwproxy.myco.com`
- プロキシ・ポート : 80
- プロキシ・レルム : Authorized Proxy Users Only

この場合、プロキシ認証用のキーは、次のようになります。

- ユーザー : proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-u
- パスワード : proxy-wwwproxy.myco.com-80-Authorized Proxy Users Only-p

WebDAV 認証用のキーは、次のようになります。

- ユーザー : myserver.myco.com-443-Authorized WebDAV Users Only-u
- パスワード : myserver.myco.com-443-Authorized WebDAV Users Only-p

ユーザー名とパスワードは、*`$ORACLE_HOME`* の `bin` ディレクトリにある `mkstore` コマンドを使用して、Oracle ウォレットに入力されます。Oracle ウォレットの作成と変更には、そのウォレットの作成時に指定したパスワードが必要です。ただし、ウォレットは、ユーザー名とパスワードを参照する実行時にパスワードが要求されないように構築されています。したがって、この機密データを保護するためには、ファイル・システム権限を使用してアクセスを制限する必要があります。アクセス権の付与は、実行時のウォレットにアクセスする必要があるユーザーのみに限定する必要があります。デフォルトでは、`mkstore` コマンドによって、権限が制限されたウォレットが作成されます。

次のコマンドでは、ウォレットが /wallets ディレクトリに作成され、ユーザー名とパスワードが格納されます。この例のユーザー名とパスワードは、proxyUser、proxyPassword、webdavUser および webdavPassword です。

```
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
```

各コマンドでは、ウォレットのパスワード入力を求めるプロンプトが表示され、必要な場合は、そのウォレットのディレクトリ（rules_wallet がディレクトリです）が作成されます。

次のコマンドでは、mkstore コマンドの様々な機能をリストした使用方法のメッセージが出力されます。

```
mkstore -help
```

B.3 ファイル・リポジトリの使用

この項では、ファイル・リポジトリの設定と使用について説明します。

B.3.1 ファイル・リポジトリの設定

Oracle Business Rules には、ディクショナリを含まない空のファイル・リポジトリが用意されています。このファイル・リポジトリは、emptyFileRepository という名前で、*$ORACLE_HOME*/rules/lib ディレクトリ内にあります。

新規のファイル・リポジトリを設定するには、emptyFileRepository ファイルをコピーして名前を変更します。次に、このファイル名と場所を「リポジトリ」タブにある「接続」ページで指定します（2.4.1 項「Rule Author リポジトリへの接続」を参照）。

新規のファイル・リポジトリを作成した後は、そのファイル・リポジトリに接続し、リポジトリ内にディクショナリを作成して保存できます。

選択されているリポジトリ・タイプが「ファイル」である場合に、「リポジトリ」タブにある「接続」ページで**「作成」**をクリックして新規のファイル・リポジトリを作成することもできます。既存のリポジトリ・パスを入力し、**「作成」**をクリックした場合、この作成は**「接続」**をクリックした場合のように動作し、既存のリポジトリに接続します。

B.3.2 ファイル・リポジトリの更新とテンポラリ・ファイル

SDK によって、RepositoryConnection インタフェースが起動し、リポジトリのコンテンツが更新されると、次の操作が実行されます。

1. テンポラリ・ファイルが作成され、更新されたコンテンツが格納されます。このテンポラリ・ファイルが必要な理由は、JAR ファイルのリライト処理で、現行リポジトリから未読エントリが読み込まれる場合があるためです。また、新規コンテンツの書込みに何か問題が起きた場合の安全対策にもなります。テンポラリ・ファイルは、File.createTempFile メソッドを使用して作成します。リポジトリの名前が 3 文字未満の場合は、「_tmp_」が追加されます。File.createTempFile メソッドでは、リポジトリの名前が 3 文字以上である必要があります。Sun JDK では、名前に数字を追加しますが、他の JVM 動作は異なる場合があります。このファイル名の拡張子は「.tmp」で、ファイルは既存のリポジトリと同じディレクトリに作成されます。要約すると、myRepository というリポジトリのテンポラリ・ファイル名は myRepository65146.tmp になり、rr というリポジトリのテンポラリ・ファイル名は rr_tmp_65147.tmp になります。
2. コンテンツがテンポラリ・ファイルに書き込まれます。
3. 既存のリポジトリの名前は、既存のリポジトリ名に「_o_r_i_g_」とミリ秒単位の現行時間（UTC）が追加された名前に変更されます。
4. テンポラリ・ファイルは、リポジトリの名前（myRepository など）に変更されます。
5. 名前が変更されたリポジトリ（前のコンテンツが含まれている）は削除されます。

このプロセスでエラーが発生した場合は、クリーン・アップが試みられます。以前に作成されたテンポラリ・ファイルが現在も存在する場合は、削除が試みられます。また、既存のリポジトリの名前が変更された場合は、元の名前の復元が試みられます。

テンポラリ・ファイルが残されている場合は、更新を試みる前にそのファイル・リポジトリが存在している必要があります。コンテンツの状態が不明なテンポラリ・ファイルは削除してください。

改名したリポジトリ・ファイルはあるが、元のリポジトリ・ファイルがすでにない場合、この改名したリポジトリ・ファイルには、更新前のコンテンツが含まれているため、手動で復元する必要があります（つまり、名前を変更するか、または改名したファイルをコピーして正しい名前に戻します）。

B.4 リポジトリの高可用性

構成した後の WebDAV リポジトリは、バックアップ・プロセスとリカバリ・プロセスの対象となるように、OracleAS Recovery Manager の構成に追加する必要があります。

OracleAS Recovery Manager の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

> **関連項目：** Oracle Business Rules および高可用性の詳細は、『Oracle Application Server 高可用性ガイド』を参照してください。

C

Oracle Business Rules に関するよくある質問

この付録では、Oracle Business Rules に関するよくある質問を紹介します。質問に対する回答は、次の各カテゴリに従って提供されます。

- ルール運用に関する質問
- Oracle Business Rules を使用するには、どのような JAR ファイルが必要ですか。
- Rule Author を Oracle 以外のコンテナにデプロイするにはどうすればよいですか。
- RuleSession では並行性と同期化はどのように処理されますか。
- Oracle Business Rules のランタイム・パフォーマンスを向上させるにはどうすればよいですか。
- RL Language のクロス積の正しい使用方法を教えてください。
- URL を使用してディクショナリのルールにアクセスするにはどうすればよいですか。
- Oracle Business Rules ではプロパティ変更リスナーをどのように使用しますか。
- Oracle Application Server での Single Sign-On タイムアウトを変更するにはどうすればよいですか。
- Oracle Business Rules に高可用性は備わっていますか。

C.1 ルール運用に関する質問

この項では、Oracle Business Rules のルール運用のセマンティクスに関連した質問について説明します。

C.1.1 ルール・アクションのファクトの状態が、ルール条件と矛盾しているのはなぜでしょうか。

ルールのアクティブ化時間と起動（実行）時間のと間に、オブジェクトが変更されたため、そのオブジェクトは、Rules Engine で再度アサートされませんでした。

オブジェクト（Java または RL）は、ルール評価で使用する前に、ファクトとして Rules Engine でアサートされる必要があります。ファクトとしてアサートされたオブジェクトが、ルールのアクションまたは Rules Engine 外部の何か（多くの場合、アプリケーション）によって変更された場合、そのオブジェクトは、現在のオブジェクト状態を Rules Engine とルール評価に反映するために、Rules Engine で再度アサートされる必要があります。この操作を実行しない場合、アプリケーションは Rules Engine と矛盾した状態になり、予期しない状態が発生する可能性があります。

`PropertyChangeListener` をサポートするために Java Bean を記述すると、Bean プロパティの更新時に、Rules Engine が一貫性のある状態を自動的に保持できるようになります。詳細は、1.3.4.1 項「Java ファクト・タイプ定義」を参照してください。

このルールの例外は、評価されないコンテンツのオブジェクトの場合です。つまり、Rules Engine には、そのオブジェクトのメソッドまたはプロパティをテストまたはアクセスするルールがありません。このような場合の例には、ルール評価の結果を累計するために使用するオブジェクトがあります。

> **注意：** ファクトのコレクションから合計を算出するルールがあると仮定します。値が累積されているファクトを再アサートすると、その合計がファクトに含まれて正しい結果が得られません。また、合計を算出するルールも再アサートする必要があります。

C.1.2 変更した Java オブジェクトをファクトとしてアサートしましたが、ルールが起動しません。なぜでしょうか。

そのオブジェクトは、Rules Engine で再アサートする必要があります。Rules Engine では、ルール条件を再評価せず、ルールもアクティブ化していません。詳細は、C.1.1 項を参照してください。

C.1.3 Oracle Business Rules RL Language と Java の相違は何ですか。

『Oracle Business Rules ランゲージ・リファレンス』の付録 A を参照してください。

C.1.4 Rules SDK を使用して式に NULL を含めるにはどうすればよいですか。

Rules SDK では、文字列値に自動的に引用符が追加されます。引用符なしの NULL 値など、引用符なしの値を式に含めるには、拡張式を指定します。式タイプ FORM_LITERAL では、文字列値が常に引用符で囲まれます。式タイプ FORM_ADVANCED を使用した場合は、式は引用符で囲まれません。

たとえば、次の Rules SDK コードには NULL が含まれています。

```
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");
```

FORM_ADVANCED を使用しないと、次のエラーが表示されることがあります。

```
SEVERE: RUL-01815: key または値 put （ LiteralValue、null ）に NULL 値は許可されません

java.lang.NullPointerException: RUL-01815: key または値 put （ LiteralValue、null ）に NULL 値は許可されません
```

C.2 Oracle Business Rules を使用するには、どのような JAR ファイルが必要ですか。

Oracle Business Rules のサポートには、表 C-1 に示した JAR ファイルが必要です。すべてのパスは、*$ORACLE_HOME* に対する相対パスです。

表 C-1 Oracle Business Rules 必要な JAR ファイル

JAR ファイル	説明
rules/lib/rl.jar	Oracle Business Rules Rules Engine ライブラリ。インスタンス化および Rules Engine との対話に使用する Java API です。
rules/lib/rl_dms.jar	Rules Engine ダイナミック・モニタリング・サービス（DMS）のサポート。RuleSession に対して DMS が使用可能な場合、このファイルは必須です。
rules/lib/rulesdk.jar	Oracle Business Rules SDK。Rule Author をプログラムで使用するための Java API です。
rules/lib/webdavrc.jar	WebDAV リポジトリのサポート用の Oracle Business Rules SDK ライブラリ。WebDAV リポジトリで SDK を使用する場合、このファイルは必須です。
rules/lib/jr_dav.jar	WebDAV クライアント・ライブラリ。WebDAV リポジトリで SDK を使用する場合、このファイルは必須です。
jlib/oraclepki.jar	WebDAV リポジトリなどのリポジトリで認証をサポートする場合、このファイルは必須です。
jlib/ojpse.jar	WebDAV リポジトリなどのリポジトリで認証をサポートする場合、このファイルは必須です。
rules/lib/jsr94.jar	標準 JSR-94 ライブラリ。
rules/lib/jsr94_obr.jar	Oracle Business Rules JSR-94 の実装。
lib/xml.jar	Rules SDK で必須のファイルです。
lib/xmlparserv2.jar	Rules SDK で必須のファイルです。
j2ee/home/lib/http_client.jar	WebDAV リポジトリを使用する場合、このファイルは必須です。

C.3 Rule Author を Oracle 以外のコンテナにデプロイするにはどうすればよいですか。

この項では、Rule Author と Rule Author のオンライン・ヘルプを Oracle 以外のコンテナにデプロイする手順を示します。

この項の内容は次のとおりです。

- WebSphere V6.1（WAS V6.1）への Rule Author のデプロイ
- WebLogic Server への Rule Author のデプロイ
- JBoss 4.0 への Rule Author のデプロイ

C.3.1 WebSphere V6.1（WAS V6.1）への Rule Author のデプロイ

Rule Author を WebSphere V6.1 にデプロイするには、まず Oracle Application Server に付属の OracleAS Companion CD のインストール・ディスクを見つけます。OracleAS Companion CD の Disk 2 には、サポートされている Oracle 以外の各コンテナ用に 2 つの .ear ファイルが含まれています。WebSphere 用の .ear ファイルは、次のディレクトリにあります。

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

> **注意：** WebSphere では、Rule Author オンライン・ヘルプは使用できません。

WebSphere のインストール環境に管理セキュリティを構成することをお薦めします。管理セキュリティの設定方法については、WebSphere のマニュアルを参照してください。

Rule Author のインストールの一環として、認証および認可用に適切なユーザーおよびグループ名を構成する必要があります。次の手順では、スタンドアロン・カスタム・レジストリを使用して Websphere WAS V6.1 のユーザーとパスワードを設定する方法の例のみを示します。Rule Author の本番使用では、熟練者によってセキュリティが構成されている必要があります。データベースや LDAP など、さらに詳しい方法の例については、該当する IBM マニュアルを参照してください。

Oracle Business Rules Rule Author を WebSphere V6.1 にデプロイする手順は、次のとおりです。

1. WebSphere サーバーを起動します。

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

2. 次の URL を使用して、WebSphere 管理コンソールにログインします。

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

   > **注意：** グローバル・セキュリティを有効にした場合は、管理グループに属するユーザー名で管理コンソールにログインする必要があります。

   a. ナビゲーション・ツリーで「Security」ノードをクリックして展開します。

   b. 「Secure administration, applications, and infrastructure」ノードをクリックします。

   c. 「User account repository」グループ・ボックスで、「Available realm definitions」から「Standalone custom registry」を選択し、「Set as current」をクリックします。

   d. **「Apply」**をクリックします。

   e. ページの上部にある「messages」ボックスで「Save」リンクをクリックします。

f. サーバーのファイル・システムで、WebSphere を実行するオペレーティング・システム・ユーザーのみがアクセスできるディレクトリを選択し、そのディレクトリ内にグループ・ファイルおよびユーザー・ファイルを作成します。これによって 2 人のユーザーが作成されます。1 人は、管理コンソールにログインする特殊権限を持つ AdminGroup グループに属する adminUser、もう 1 人は、Rule Author アプリケーションへのアクセス権限が付与される ruleUser1 です。

次に、グループ・ファイル groups.prop の内容の例を示します。

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

グループ・ファイルの各行には、次のものが含まれます。

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

次に、ユーザー・ファイル users.prop の内容の例を示します。

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

ユーザー・ファイルの各行には、次のものが含まれます。

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

これにより、adminUser および ruleUser1 のパスワードがどちらも welcome1 に設定されます。

g. 「User account repository」グループ・ボックスで、「Available realm definitions」から「Standalone custom registry」を選択し、「Configure」をクリックします。

h. 「Custom properties」リンクをクリックして、ファイルの場所を指定する 2 つのプロパティを設定します。

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

i. これらの 2 つのプロパティを構成した後、上部の「Standalone custom registry」リンクをクリックして、前のページに戻ります。

j. 「Primary administrative user name」を「adminUser」に設定します。

k. 「Server identity that is stored in the repository」ラジオ・ボタンを選択します。

l. users.prop ファイルで定義済であるサーバー・ユーザー ID（adminUser）およびサーバー・パスワード（welcome1）を、同名のフィールドに入力します。ユーザー名とサーバー・パスワードは、前に指定した（手順 d を参照）ユーザー・ファイルとグループ・ファイルに含まれている必要があります。

m. 「Custom registry class name」フィールドに「com.ibm.websphere.security.FileRegistrySample」と入力します（これはデフォルト値である必要があります）。

n. 「OK」をクリックします。

o. 「Enable application security」ボックスを選択します。

p. 「Use Java 2 security」ボックスが選択されている場合は解除します。場合によっては、このボックスは前の手順でボックスを選択すると自動的に選択されるため、選択を解除する必要があります。

q. 「Apply」ボタンをクリックします。

r. ページの上部にある「messages」ボックスで「Save」をクリックします。

3. 左列の「Servers」ノードをクリックして展開し、「Application servers」をクリックします。Rule Author をデプロイするサーバーをクリックします。
4. コマンドラインから `stopServer` および `startServer` コマンドを使用してサーバーを停止し、再起動します。
5. 左側のナビゲーション・パネルで「Application」→「Install New Application Node」をクリックします。ruleauthor_websphere.ear へのパスを指定し、「Next」をクリックする必要があります。

   「ApplicationName」フィールドを「Rule Author」（空白あり）から「RuleAuthor」（空白なし）に変更します。「Next」をクリックしてから、続く 2 つの画面で「Next」をクリックします。最後に「Finish」をクリックします。

   - 左側のパネルで「Environment」をクリックします。
   - 「Shared Libraries」をクリックします。
   - 「Scope」メニューから目的の有効範囲を選択します。
   - 「new」をクリックします。
   - 名前として「Oracle XDK」と入力します。
   - 「Classpath」に、「xml.jar」および「xmlparserv2.jar」ファイルへの絶対パスを入力します。これらのファイルは、デプロイした RuleAuthor.ear アプリケーションの格納場所にあります。（「find」または「search」を使用して検索）。パスは行ごとに 1 つずつ入力し、セパレータ（「:」など）は使用しません。
   - 「OK」ボタンをクリックします。
   - 「Save」をクリックします。
6. 「Applications」領域で、「Enterpirse Applications」をクリックしてから「RuleAuthor」リンクをクリックします。
   - 「References」見出しの下で、「Shared library references」をクリックします。
   - 「RuleAuthor」エントリのチェック・ボックスを選択し、「Reference shared libraries」ボタンをクリックします。
   - 「Available」リストで「Oracle XDK」エントリを選択し、これを「Selected」リストに移動します。
   - 「OK」をクリックします。
   - 次のページで「OK」をクリックします。
7. 「Security role to user/group mapping」をクリックします。
8. 「Security role to user/group mapping」画面で「Select」列のチェック・ボックスを選択し、「Look up groups」ボタンをクリックします。
9. 「Search」ボタンをクリックして選択可能なグループを表示し、ユーザー「group1」を「Available」ボックスから「Selected」ボックスに移動します。「OK」をクリックします。
10. 戻ったページで、「Save」をクリックします。
11. WebSphere コンソールをログアウトします。
12. コマンドラインから WebSphere インスタンスを停止し、再起動します。
13. ブラウザで次の URL（デフォルト・ポートは 9080）を指します。

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

手順 2 で RuleAdministrator ロールをマップした場合は、ページ上の任意のリンクをクリックすると Rule Author ログイン・ページが表示されます。RuleAdministrator ロールを正しくマップしていない場合は、アプリケーションで直接「Repository Connection」ページが表示されます。

Oracle ウォレットを使用するパスワード保護された WebDAV リポジトリへのアクセスは、現在 WebSphere では機能しません。WebDAV リポジトリへのパスワード保護されたアクセスは、直接指定されたユーザー名とパスワードを使用した場合のみ可能です。

注意： XML スキーマをインポートするには、まず xml.jar ファイルおよび xmlparserv2.jar ファイル（インストール済の Rule Author アプリケーション・ディレクトリ内に格納）を、Java ファクト・クラスパスに追加します。XML スキーマから JAXB コンパイラにより生成される Java クラス・ファイルは、これらの JAR ファイルで指定された複数のクラスに依存します。Oracle 以外のコンテナでは、これらのクラスはデフォルトで Java ファクト・クラスパスにないため、手動で追加する必要があります。

C.3.2 WebLogic Server への Rule Author のデプロイ

WebLogic Server 9.1 に Oracle Business Rules Rule Author および Rule Author のオンライン・ヘルプをデプロイするには、まず Oracle Application Server に付属の OracleAS Companion CD のインストール・ディスクを見つける必要があります。OracleAS Companion CD の Disk 2 には、サポートされている Oracle 以外の各コンテナ用に 2 つの .ear ファイルが含まれています。

展開した ear ファイルを使用して Oracle Business Rules Rule Author を WebLogic Server 9.1 にデプロイする手順は、次のとおりです。

注意： OracleAS Companion CD の Disk 2 上の weblogic .ear ファイル、`ruleauthor_weblogic.ear` および `rulehelp_weblogic.ear` を、次のディレクトリで見つけます。

```
Disk2/rules/webapps/weblogic/
```

---

1. ディレクトリ `ruleauthor` を、*<bea home>* の任意の場所に作成します。*bea home* は、WebLogic Server 9.1 のホーム・ディレクトリを示します。
2. このディレクトリへ移動し、OracleAS Companion CD の Disk 2 上にある `ruleauthor_weblogic.ear` を、このディレクトリ内で次のように展開します。

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

3. WebLogic Server を次のように起動します。

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

4. ブラウザで WebLogic Server 管理コンソールを指してログインします。次に例を示します。

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

5. 左側のパネルで「Deployments」をクリックします。
6. 左側のパネルで「Lock & Edit」ボタンをクリックします。
7. 「Install」ボタンをクリックします。
8. `webapp` ディレクトリを見つけるまで、ファイル・ブラウザをナビゲートします。このディレクトリのラジオ・ボタンを選択します。指示に従ってアプリケーションをデプロイします。
9. 左側のパネルで「Activate Changes」ボタンをクリックします。
10. 左側のパネルで「Lock & Edit」ボタンをクリックします。
11. 左側のパネルで「Security Realms」をクリックします。
12. 「myrealm」をクリックします。

13. 「Users and Groups」タブをクリックします。
14. 「Groups」サブタブをクリックします。
15. 「New」ボタンをクリックします。新規グループ rule-administrators を作成します。
16. 「Users」サブタブをクリックします。
17. 「New」ボタンをクリックします。新規ユーザーを作成します。名前を決めて入力します。たとえば、ruleadmin と入力し、この新規ユーザーのパスワードを入力して、「Save」をクリックします。「Groups」タブをクリックします。シャトルを使用して、このユーザーを rule-administrators グループに割り当てます。完了したら、「Save」をクリックします。
18. 「Release Configuration」ボタンをクリックします。
19. 「Deployments」ページに進んでアプリケーションを起動します。
20. コンソールからログアウトして、ブラウザを閉じます。
21. アプリケーションをテストするには、ブラウザを開いて次の URL に進みます。

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

Rule Author のヘルプをデプロイするには、手順（1 ～ 7）に従って rulehelp_weblogic.ear をデプロイします。ヘルプ・ページをテストするには、Rule Author ページのヘルプ・リンクをクリックします。

注意： XML ファクトのインポートと WebDAV リポジトリ・アクセスの使用を有効にするには、コンテナ・クラスパスに Oracle XDK クラスを追加します。これには、xml.jar と xmlparserv2.jar を CLASSPATH 環境変数パスに追加して、実行中のドメインの startWebLogic.sh を更新します（Windows の場合、更新する必要のあるファイルの名前は startWebLogic.cmd です）。

C.3.3 JBoss 4.0 への Rule Author のデプロイ

JBoss に Rule Author および Rule Author のオンライン・ヘルプをデプロイするには、まず Oracle Application Server に付属の OracleAS Companion CD のインストール・ディスクを見つける必要があります。OracleAS Companion CD の Disk 2 には、サポートされている Oracle 以外の各コンテナ用に 2 つの .ear ファイルが含まれています。

注意 1: Rule Author の XML スキーマのインポート機能を使用する場合は、JBoss を実行する環境で、ファイル xml.jar および xmlparserv2.jar の場所を環境変数 JBOSS_CLASSPATH に追加する必要があります。これらのクラスは、ORACLE_HOME/lib で使用可能です。

注意 2: JBoss のもとでは、XML 解析機能に互換性がないため、WebDAV リポジトリにアクセスできません。

Rule Author を JBoss 4.0 にデプロイする手順は、次のとおりです。

注意： OracleAS Companion CD の Disk 2 上の JBoss .ear ファイル、ruleauthor_jboss.ear および rulehelp_jboss.ear を、次のディレクトリで見つけます。

```
Disk2/rules/webapps/jboss/
```

---

1. JDK 1.5 が環境にインストールされていることを確認します。JBoss は、JDK 1.4 ではうまく動作しません。
2. ruleauthor_jboss.ear を <*jboss home*>/server/default/deploy ディレクトリに挿入します。
3. 次の jars を <*jboss home*>/server/default/lib ディレクトリに挿入します。これらは、ruleauthor_jboss.ear ファイルを一時ディレクトリで展開することによって取得できます。

```
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. <*jboss home*>/server/default/conf/login-config.xml を編集して、次のセクションを追加します。

```
<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. ユーザー（ruleauthor-users.properties）およびロール（ruleauthor-roles.properties）の各ファイルを次のディレクトリに追加します。

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

   ruleauthor-users.properties ファイルの内容の例：

   ```
   ra=oraclerule
   ```

   ruleauthor-roles.properties ファイルの内容の例：

   ```
   ra=RuleAdministrator
   ```

   この例では、パスワード oraclerule を持ち、ロール RuleAdministrator に属するユーザー名 ra を作成しています。

6. 次のコマンドを使用して JBoss を起動します。

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

正常にデプロイされると、次のような出力が表示されます。

```
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. ブラウザで次のサイトを参照します。

   http://<host>:8080/ruleauthor

   手順 2 と 3 で RuleAdministator ロールをマッピングした場合は、Rule Author のログイン・ページが表示されます。

Rule Author のヘルプをデプロイするには、rulehelp_jboss.ear をデプロイ・ディレクトリにドロップします。ヘルプ・ページをテストするには、Rule Author ページのヘルプ・リンクをクリックします。

C.4 RuleSession では並行性と同期化はどのように処理されますか。

Oracle Business Rules RuleSession オブジェクトに対するメソッド・コールは、複数のスレッドによるコールによって、RuleSession レベルでの例外が発生しないようにスレッド・セーフになっています。ただし、メソッドの実行に排他性やトランザクション保証はありません。Rules Engine の最低レベルの run メソッドは同期化されるため、共有 RuleSession を持つ 2 つのスレッドが同時に run を実行できません。run への一方のコールは、もう一方のコールが終了するまで待機する必要があります。

RL 関数は、デフォルトでは同期化されません。Java メソッドと同様、RL 関数は並行して実行可能であり、プログラマは同期化されたブロックを使用して共有データ（たとえば、結果データを含む HashMap）へのアクセスを保護する必要があります。

一連のアクションがトランザクション形式のように実行されるようにするには、共有オブジェクト関連を同期化する必要があります。ユーザーは、RuleSession オブジェクト関連は同期化しないでください。これは、RuleSession メソッドをコールするときに出される例外が、RuleSession オブジェクトの破棄を必要とする場合があるためです。

RuleSession オブジェクトを Oracle Business Rules で使用するほとんどの場合、各スレッドまたはサーブレット・インスタンスは、ローカルな RuleSession オブジェクトを作成して使用する必要があります。この使用パターンは、この方法で JDBC 接続を使用した場合とおおよそ似ています。

次の例は、共有 RuleSession オブジェクトを使用する方法を示しています。

Thread-1 が次のものを含む場合 :

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

Thread-2 が次のものを含む場合 :

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

この場合、2 つのスレッドの実行は、例 C-1 で示すように進むことがあります。

例 C-1 Oracle Business Rules における共有 RuleSession オブジェクトの使用

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

例 C-1 では、Thread-1 がアサートした 2 つのファクトが両方とも、run へのコール中に RuleSession に存在することはありません。また、1 つのスレッドのみが run メソッドをコールすることに注意してください。複数のスレッドが共有 RuleSession に対して run をコールできる設計はお薦めしません。この使用パターンによって、非常に見つけにくい不具合が作成される可能性があり、パフォーマンス上の利点はありません。

共有 RuleSession オブジェクトに対するすべてのアクセスを同期化して、意図した動作を保証する必要があります。ただし、RuleSession インスタンスが例外をスローして回復不能になることがあるため、このオブジェクトを同期化オブジェクトとしては使用しないでください。かわりに、別の共有オブジェクトを同期点として使用します。

RuleSession で使用する共有サーバー・プロセスのプロデューサ / コンシューマ・モデルを想定できます。このモデルでは、複数のスレッドが共有 RuleSession に対してファクトをアサートし、1 つのスレッドが定期的に run をコールし、結果を読み取ってそれらを出力します。これにより、スレッドの対立が生じる可能性はなくなります。これは、2 つのコード・セグメントを順次実行する必要があり、混在させることができないためです。たとえば、例 C-2、例 C-3 および例 C-4 にあるような、共有オブジェクトに関するコード、プロデューサ・コードおよびコンシューマ・コードです。

例 C-2 RuleSession の共有オブジェクト

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

例 C-3 RuleSession のプロデューサ・コード

```
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;
}
```

例 C-4 RuleSession のコンシューマ・コード

```
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;
    }
  }
}
```

注意： 複数のスレッドが RuleSession オブジェクトを共有している場合、複数のスレッドが run メソッドをコールすることはお薦めしません。

C.5 Oracle Business Rules のランタイム・パフォーマンスを向上させるにはどうすればよいですか。

RuleSession オブジェクトを使用するアプリケーションのパフォーマンスを向上させるには、リポジトリを開いて RL コードを生成する回数を最小限に抑えるようにする必要があります。したがって、作成した各 RuleSession インスタンスのリポジトリを開いて読み取ることを回避します。これにより、ほとんどの場合、プログラムのパフォーマンスは向上します。

生成された RL コードを含む String オブジェクトを共有できるため、すべてのアプリケーション・インスタンスがこれを使用して、ルールをプライベートな RuleSession オブジェクトにロードできます。

アプリケーションでさらにパフォーマンスを調整する必要がある場合は、プーリング・メカニズムを使用して、一連の RuleSession オブジェクトを共有できます。この場合、共通プールは RuleSession オブジェクトを保持し、スレッドはプールからオブジェクトを取得して、一連の操作にこのオブジェクトを使用して、オブジェクトをプールに戻すことができます。例外をスローする RuleSession オブジェクトは再使用できません。例外が RuleSession によってスローされた場合、プールはこのインスタンスを破棄して新しい RuleSession インスタンスと置換する必要があります。

プールされた RuleSession オブジェクトを使用する場合は、次の問題に注意してください。

- ルール・アクションが、外部リソースへのコールを含めて、入力または出力を行わない場合、プールのサイズをマシンの CPU の数より大きくする必要はありません。
- RuleSession の状態は、再使用の前にリストアする必要があります。RL reset() 関数を使用すると便利です。reset() で RuleSession 内の非 final グローバル変数に対してイニシャライザが再実行されるようなファクトには、特に注意してください。これは、初期ファクトを再アサートするためのフックとして使用される場合があります。
- 例外をスローする RuleSession は再使用しないでください。これは、例外をスローする RuleSession インスタンスの状態を確実にリストアすることが困難であるためです。

使用するアプリケーションの計算要件によっては、プーリングによるパフォーマンスの向上がほとんどみられない場合があります。RuleSession オブジェクトの新規作成と RL コードのロードにかかる計算コストがパフォーマンスのボトルネックになることはほとんどありません。パフォーマンスを監視して、それが許容できる範囲内であるかどうかを確認してから、RuleSession オブジェクトのプーリングを使用してパフォーマンスを向上させるようにしてください。

Oracle Java Object Cache には、非常に簡単なプーリングのメカニズムが備わっています。ただし、reset() をコールして RuleSession オブジェクトの破棄と再作成を必要とする例外があるかどうかをチェックするためのレイヤーを追加する必要があります。

> **関連項目：**『Oracle Containers for J2EE サービス・ガイド』の Java Object Cache を参照してください。

C.6 RL Language のクロス積の正しい使用方法を教えてください。

ファクトのクロス積を使用した場合、RL Language のランタイム動作で極端な結果が生成されることがあります。

例 C-5 の RL Language コードについて考えます。

例 C-5 ファクト F を使用するクロス積

```
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();
```

例 C-5 でいくつの行が出力されるか考えます。ファクト `F1` とファクト `F2` は同じインスタンスであり、各オペランドで比較されるため、答えは 4 行です。

したがって、例 C-5 では次の出力が生成されます。

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

3 番目の `F` のある同じ例を使用すると（たとえば、`assert(new F(i:3));`）、9 つの行が出力されます。また同時に、3 番目の語句 `&& fact F F3` が追加されると、27 の行が出力されます。

個別のファクトについてすべての組合せと順序を見つけようとすると、例 C-6 に示すように、テスト内に追加語句が必要になります。

例 C-6 ファクト F のすべての組合せの検索

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

例 C-6 のコードでは、次の出力が生成されます。

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

ファクトのすべての組合せを検出する最も簡単な方法は、最速ではなく順序も不同ですが、例 C-7 に示したコードを使用するものです。

例 C-7 ファクト F の組合せの検索

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

例 C-7 に示した関数 `id()` をテスト・パターンで実行すると直接比較よりも時間がかかるため、各オブジェクトの固有の値に対してテストする方法が最も速い方法になります。たとえば、整数値プロパティ oid を、クラスの各インスタンスの固有値を割り当てられたクラスに追加できます。

例 C-8 は、oid 値を使用する同じルールを示しています。

例 C-8 高速の完全比較

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

この問題は、例 C-9 で示した関数を使用して、Oracle Rules Engine からすべての重複するファクトを削除しようとした場合にも生じることがあります。

例 C-9 重複するファクトを取り消す不正な例

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

ただし、このルールは重複だけではなくタイプ F のすべてのファクトを削除します。これは、F1 および F2 が同じファクト・インスタンスである場合があるためです。例 C-10 は、このルールの正しいバージョンを示しています。

例 C-10 重複するファクトを取り消す修正された例

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

C.7 URL を使用してディクショナリのルールにアクセスするにはどうすればよいですか。

Rule Author では、URL にディクショナリとルール名を使用して、URL からルールに直接アクセスできます。ユーザーは、Rule Author ログイン・ページまたは Oracle Single Sign-On を使用して Rule Author にログインした後は、Rule Author で URL を使用してルールセットまたはルール・ページを表示できます。

URL の形式は次のとおりです。表 C-2 は、パラメータの説明を示します。

```
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>
```

たとえば、次の URL を使用すると、ルールセット rs2 のルール checkCredit を単一の URL で表示できます。

```
http://localhost:8888/ruleauthor/ConnectRepos.uix?event=dlConnect&ReposType=File&oracle
.rules.sdk.store.jar.path=C:¥jartest¥test1.jar&Dict=c233&Version=v1&RuleSet=rs2&rule=ch
eckCredit
```

表 C-2 URL を使用してルールにアクセスするための URL パラメータ

URL コンポーネント	説明
host_name	Rule Author アプリケーションが動作中のホストを指定します。
port_number	Rule Author アプリケーションが動作中のポート番号を指定します。
repos_prop i	Oracle Business Rules SDK によって定義されたリポジトリ接続プロパティ *i* を示します。追加の詳細は、Javadoc を参照してください。 ファイル・タイプ・リポジトリについては、oracle.rules.sdk.store.jar.Constants.java を参照してください。 WebDAV については、oracle.rules.sdk.store.webdav.Constants.java を参照してください。
repos_value val	プロパティ *i* のリポジトリ接続値 *val* を示します。
dict_version	ディクショナリ名を指定します。
ruleset_name	（オプション）ルールセット名を指定します。*<ruleset_name>* または *<rule_name>* のいずれも指定しない場合は、「ルールセット・サマリー」ページが表示されます。
rule_name	（オプション）ルール名を指定します。*<ruleset_name>* を設定して、*<rule_name>* を指定しない場合は、「ルールセット」ページが表示されます。

C.8 Oracle Business Rules ではプロパティ変更リスナーをどのように使用しますか。

Oracle Rules Engine は、Java PropertyChangeListener 設計パターンをサポートしています。これにより、PropertyChangeSupport クラスを使用する Java ファクトのインスタンスは、プロパティ値が変更されたときに自動的に Oracle Rules Engine に通知できます。Java ファクトは、Oracle Rules Engine によって使用されるこのパターンを実装するために必要とされません。

通常、以前に Oracle Rules Engine に対してアサートされた Java オブジェクトのプロパティの値に変更が加えられた場合は、ルールを新しいプロパティ値によって再評価するために、オブジェクトを再アサートする必要があります。PropertyChangeEvents を起動するプロパティの場合、これらのプロパティの値を変更すると、値が変更されて、ファクトが Oracle Rules Engine に対して再アサートされます。

PropertyChangeListener 設計パターンをクラスに実装する手順は次のとおりです。

1. このパッケージをクラスにインポートします。

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

2. プライベート・メンバー変数をクラスに追加します。

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

3. コンストラクタで、Property Change Support オブジェクトを作成します。

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

4. 次の各 setter に対して、firePropertyChange へのコールを追加します。

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

5. addPropertyChangeListener メソッドを実装します（m_pcs に委任します）。

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

6. removePropertyChangeListener メソッドを実装します（m_pcs に委任します）。

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

変更されたオブジェクトを常に明示的に再アサートするか、または PropertyChangeListener 設計パターンを実装するか、どちらにアプリケーションを設計するかを決定する場合は、次の項目について考慮してください。

- 変更されたオブジェクトを明示的に再アサートすると、ユーザーは、一部のプロパティ変更をグループ化して、これらを一度にルールに表示できます。これは、並行スレッドがルールを実行しているときにもっとも便利であり、ルールにはプロパティ変更の完全なグループのみが表示されます。
- 明示的なアサートによって、複数のプロパティが変更された場合に、ルール再評価の計算コストが削減されます。複数のプロパティが同時に変更されると、ファクト・タイプを参照するルール条件の複数の再評価が生じます。これは、各プロパティ変更イベントによってオブジェクトの再アサートが生じるためです。PropertyChangeListener パターンのかわりに明示的なアサートを使用すると、この余分な計算コストを除去できます。
- 明示的なアサートは、ルールがその条件でもテストされるファクトを変更する場合に必要ですが、ガード条件プロパティが設定される前に PropertyChangeListener によって起動される自動再アサートでは、ルールがそれ自体を無限に再起動します。
- 明示的なアサートは、RL ファクトと XML ファクトを変更する場合に使用する必要があります。これは、PropertyChangeListener 設計パターンをサポートするように、これらを定義できないためです。
- PropertyChangeListener 対応ファクトを使用すると、Java アプリケーションは、アプリケーションで明示的アサートを実行するように変更しなくても、プロパティ変更をルール・エンジンに伝達できます。これはつまり、オブジェクトのプロパティを変更するコードが、範囲内の RuleSession オブジェクトへの参照を持つ必要がないということでもあります。
- PropertyChangeListener サポートは、プロパティ変更後のファクトの再アサートを無視するという一般的なエラーを防止します。

C.9 Oracle Application Server での Single Sign-On タイムアウトを変更するにはどうすればよいですか。

Rule Author を含めて、アプリケーションに指定されたデフォルトの Oracle Single Sign-On タイムアウトの値がインストール環境で低すぎることがわかった場合は、このタイムアウトに指定された値を変更できます。

Single Sign-On のタイムアウトを変更する手順は、次のとおりです。

1. Application Server Control で、「クラスタ・トポロジ」ページに進みます。
2. 「管理」領域で、**「Java SSO 構成」**をクリックします。
3. 「プロパティ」領域で、**「セッション・タイムアウト」**テキスト・ボックスの値を任意のタイムアウト値に変更します。
4. **「適用」**をクリックします。

関連項目： A.2 項「Rule Author のセッション・タイムアウト」

C.10 Oracle Business Rules に高可用性は備わっていますか。

ルール・アプリケーションを高可用性アプリケーションとして実行する必要がある場合は、Oracle Business Rules および高可用性について、『Oracle Application Server 高可用性ガイド』を参照してください。

D

Oracle Business Rules のトラブルシューティング

この付録では、Oracle Business Rules の使用時に発生する問題の回避方法と解決方法について説明します。内容は次のとおりです。

- Rule Author でアクセスできないパブリック・ファクト変数
- RL 関数で使用できないグローバル変数
- JDK 1.4.2 クラスのインポート
- Firefox 上のポップアップ・ウィンドウの管理
- メソッドでの文字列データ型の使用
- データ・モデル内のクラス順序と階層の保持
- Rule Author から生成された RL の検証およびチェック
- Java パッケージ名の一部としての RL 予約語の使用
- 非表示の getter メソッドと setter メソッド
- 実行時にアサートされない XML ファクト
- Rule Author の使用時の言語の変更
- Microsoft Windows でルールセットの編集と実行を同時に行う場合にファイル・エラーが発生する理由
- 上位メソッドをサブクラスから表示できない理由
- XML スキーマを追加すると、エラー RUL-01627 が発生します。
- クライアントとサーバーが異なるロケールを使用している選択リストによる無効な RL Language の生成
- 継承されたクラスの使用時に生成される無効な RL Language

D.1 Rule Author でアクセスできないパブリック・ファクト変数

パブリック・ファクト変数は、Rule Author ではアクセスできません。たとえば、次のクラスの変数は、Oracle Business Rules RL Language ではアクセスできますが、Rule Author ではアクセスできません。

```
public class Test {
   public int i = 0;
   public String s = "string";
}
```

Test タイプのファクトの場合、Rule Author でアクセスできる変数はありません。これらの変数にアクセスするためには、次のようなメソッドを追加する必要があります。

```
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; }
```

変数 `i` は、`setI(int i)` および `getI()` の正常な動作には不要であることに注意してください。詳細は、Sun 社の Java Bean 仕様を参照してください。

D.2 RL 関数で使用できないグローバル変数

SDK（Rule Author など）から生成された RL の場合、RL 関数ではグローバル変数を直接参照できません。

この問題を回避するには、RL 関数がグローバル変数にアクセスする必要がある場合、グローバル変数をパラメータとして RL 関数に渡す必要があります。パラメータ名によって、RL 関数本体内のグローバル変数にアクセスできます。

D.3 JDK 1.4.2 クラスのインポート

JDK 1.4.2 を使用して Rule Author を実行する場合は、JDK 1.5 を使用してコンパイルした Java クラスが適切にインポートされないことに注意してください。JDK 1.5 を使用してコンパイルした Java クラスを、JDK 1.4.2 を使用して Rule Author にインポートすると、次のようなエラー・メッセージが表示されます。

```
操作を実行できません。''RUL-01527: loadClass に対する例外を受け取りました。RUL-01016: Java クラス example7.Example7 をロードできません。このクラスとすべての依存クラスがクラスパスまたはユーザー指定パスにあることを確認してください。根本的原因 : example7/Example7 (Unsupported major.minor version 49.0) ''
```

この問題を回避するには、JDK 1.5 を使用して Rule Author を実行するか、そのクラスを JDK 1.4.2 を使用して再コンパイルします。

D.4 Firefox 上のポップアップ・ウィンドウの管理

Firefox ブラウザで Rule Author を実行している場合は、多くのポップアップ・ウィンドウを閉じるために、**「OK」**、**「取消」**または**「適用」**ボタンを使用するかわりに、ウィンドウの上部隅にある「X」ボタンを使用すると、問題が発生する場合があります。

この問題を回避する簡単な方法は、ウィンドウ・コントロールを使用してポップアップ・ウィンドウを閉じるかわりに、**「OK」**、**「取消」**または**「適用」**ボタンを使用することです。また、`dom.popup_maximum` パラメータの値を変更すると、より多数のポップアップ・ウィンドウを使用できます。手順は次のとおりです。

1. URL の `about:config` を入力し、`dom.popup_maximum` パラメータを検索します。
2. 値を 10000 以上に設定します。

D.5 メソッドでの文字列データ型の使用

組込みデータ型の `String` には、メソッドが含まれていません。したがって、`x` が文字列の場合、`x.substring(1)` は、拡張式では無効となります。

この問題を回避する手順は、次のとおりです。

1. `java.lang.String` を、Java ファクト・タイプとしてデータ・モデルにインポートします。
2. このファクト・タイプの別名を指定します。デフォルトの別名は、`java_lang_String` です。
3. データ・モデルに RL ファクト・タイプまたは変数を定義する場合は、`String` のかわりにこの新規のファクト・タイプを使用します。

D.6 データ・モデル内のクラス順序と階層の保持

Rule Author で使用するクラスおよびインタフェースは、次のルールに従う必要があります。

1. クラスまたはインタフェースとそのスーパークラスを使用している場合は、最初にスーパークラスを宣言する必要があります。宣言しない場合は、生成された RL プログラムによって、次のように例外がスローされます。

   ```
   "FactClassException: fact class for 'pkg.Parent' should be declared earlier
   in rule session".
   ```

2. クラスまたはインタフェースを使用している場合、宣言できるのは、そのスーパークラスまたは実装されたインタフェースの 1 つのみです。複数のインタフェースを宣言すると、生成された RL Language プログラムによって、次のような例外がスローされます。

   ```
   MultipleInheritanceException: fact class 'pkg.Child' cannot extend both
   'pkg.ParentInterface' and 'pkg.ParentClass'
   ```

この問題を回避する手順は、次のとおりです。

1. ルールセットで使用するデータ・モデルにクラスの階層とインタフェースを指定します。
2. 階層内のクラスまたはインタフェースごとに、**「XPath アサーションのサポート」**ボックスを選択します。この操作によって、データ・モデル RL の一部として、ファクト・クラス文が適切な順序で生成されます。

D.7 Rule Author から生成された RL の検証およびチェック

Rule Author から生成された RL を検証するためには、データ・モデル内の Java クラスが、OC4J クラスパスに存在することを確認してください。同様に、XML スキーマを使用する場合は、生成された JAXB クラスがクラスパスに存在する必要があります。OC4J クラスパスの設定の詳細は、3.11 項「テスト・ルールセットの使用」を参照してください。

D.8 Java パッケージ名の一部としての RL 予約語の使用

RL Language の予約語（`mypkg.rule.com` の `rule` という語など）が Java パッケージ名の一部である場合は、無効な RL Language が生成されます。RL Language の予約語を Java パッケージ名で使用すると、次のようなエラー・メッセージが表示されます。

```
Oracle RL 1.0: syntax error PareseException: encountered 'rule' when expecting
one of: <XML_IDENTIFIER> ...<IDENTIFIER> ... "*" at line 11 column 19 in main
```

この問題に対する回避方法はありません。Java パッケージ名には RL Language の予約語を使用しないでください。

D.9 非表示の getter メソッドと setter メソッド

Rule Author では、Java Bean プロパティをサポートしているメソッドが選択リストに表示されません。表示されるのは、Bean プロパティのみです。たとえば、Y という名前のプロパティを持つ Java Bean には、少なくとも getter メソッド（`getY()`）が設定されている必要があり、setter メソッド（`setY(y-type-parm)`）も設定されている場合があります。Java ファクト・タイプを表示すると、すべてのプロパティとメソッド（プロパティを構成する getter と setter を含む）が表示されます。選択リストに表示されるのは、Java クラスのプロパティ（getter メソッドと setter メソッド以外）のみです。プロパティ表示の制御を試みる場合は、プロパティの表示可能性フラグを使用することをお薦めします。getter メソッドまたは setter メソッドを非表示としてマーク付けすると、プロパティを選択リストから削除できなくなります。

現在、この問題に対する回避方法はありません。

D.10 実行時にアサートされない XML ファクト

XML スキーマで生成したクラスの「XML ファクト」ページには、**「XPath アサーションのサポート」**ボックスが表示されます。デフォルトでは、このボックスが選択されています。このボックスの選択を解除して変更内容を保存すると、XML ファクトに、XML スタイル・アサーションの非サポートのマークが付けられます。これにより、このタイプのインスタンスとその子は、XML 文書の assertXPath へのコールによってアサートされません。

この問題に対する回避方法はありません。すべての XML ファクト・タイプが、**「XPath アサーションのサポート」**ボックスで選択されていることを確認してください。

D.11 Rule Author の使用時の言語の変更

Rule Author の表示言語を変更するには、ログアウトし、言語を変更してから、Rule Author に再度ログインする必要があります。

ログアウトしない場合、`java.lang.NullPointerException` エラーが表示される場合があります。

言語の変更に必要な手順は次のとおりです。

1. Rule Author からログアウトします。
2. ブラウザで言語を変更します。
3. Rule Author に再度ログインします。

D.12 Microsoft Windows でルールセットの編集と実行を同時に行う場合にファイル・エラーが発生する理由

Microsoft Windows オペレーティング・システムでは、あるアプリケーションで使用されているファイルは、別のアプリケーションで使用できません。これは、アプリケーションがローカル・ファイル・リポジトリから読み取ろうとしているときに、Rule Author が同じリポジトリに書き込もうとすると、エラーが発生する場合があることを意味します。関係する時間枠が小さいため、このエラーはめったに発生しません。

このタイプの障害のシグネチャは次のようになります。

```
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:265)
at
oracle.tip.tools.ide.rules.ide.jdeveloper.JDevRulesProject.saveDictionary(JDevRulesProj
ect.java:83)
```

これは、別のルールベース・アプリケーションがリポジトリの更新を試みると同時にリポジトリを読み取ることが原因で発生することがあります。操作は再試行できます。このエラーが解決しない場合は、他のアプリケーションによってリポジトリが開いたままになります。

この問題を回避するには、単に操作を再試行します。

D.13 上位メソッドをサブクラスから表示できない理由

スーパークラスのプロパティは適切な選択リストに表示されますが、上位クラスのメソッドは表示されません。

この問題に対する回避方法はありません。

D.14 XML スキーマを追加すると、エラー RUL-01627 が発生します。

スキーマを追加しようとすると、Rule Author によって次のエラーがレポートされる場合があります。

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

スキーマを追加したときにスキーマ・パス・ボックスにそのスキーマが表示されないため、リストが間違っているか、エラーが間違っていると思われます。この問題に対する回避方法は、「XML ファクト」ページをリロードしてから、スキーマを再度追加することです。

D.15 クライアントとサーバーが異なるロケールを使用している選択リストによる無効な RL Language の生成

「何も選択されていません」を示す選択リスト・エントリは変換されます。変換後の値は、RuleDictionary のロケール・セットによって制御されます。これによって Rules SDK で間違った動作を引き起こす可能性のある使用事例には次の 2 つがあります。

- `RuleDicitonary` には `setLocale` メソッドがあります。クライアント / サーバーの使用では、クライアントとサーバーが同じロケールを使用していない場合で、サーバーのロケールが英語でない場合、setLocale を使用してクライアントのロケールを設定できないため、無効な RL Language が生成されることがあります。
- サーバーのロケールが英語でない場合で、アプリケーションが `setX`（X はリストのプロパティ名）を起動しない場合、Rules SDK は選択されていないエントリを正規形（英語）に正しく設定しません。この結果、（通常、値が欠落しているために）正しくコンパイルされないエントリが生成されます。

この問題を回避する手順は、次のとおりです。

クライアントおよびサーバーが異なるロケールを使用しているか、サーバーが英語ロケールでない環境でアプリケーションが実行されている場合、そのアプリケーションは、リストが表示されている場合で、すべてのリストを設定する必要がある場合に、`RuleDictionary setLocale` を使用して、ロケールをクライアントのロケールに設定する必要があります。たとえば、Expression Form プロパティがアプリケーションによって明示的に設定されていない場合に、この問題が発生する可能性があります。

D.16 継承されたクラスの使用時に生成される無効な RL Language

場合によっては、ルールセットに対して無効な RL Language が生成されます。このコードを実行しようとすると、次のエラーがレポートされます。

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

このエラーは、継承関係を持つ 2 つの Java クラスがルールセットで使用され、子クラスが親クラスの前に使用される場合に発生します。このような使用は、ルール条件、`assert`、`retract` または `assertXPath` で発生する可能性があります。サブクラスまたはインタフェースがこれらのコンテキストのいずれかでスーパークラスより前に参照された場合、これらのコンテキストのいずれかでスーパークラスを参照すると、RL Language の解析時に `FactClassException` が発生します。

関係する各 Java クラスの `supports xpath` 属性がデータ・モデルで `false`（デフォルト）に設定されている場合は、この問題を解決するために、Rule Author は正しい RL Language、`fact class` 文を生成しません。

この問題を回避するには、関係する各 Java クラスの `supports xpath` 属性をデータ・モデルで `true` に設定する必要があります。Rule Author でこれを行うには、Java クラスを表示するときにチェック・ボックスを使用します。一部のクラスでは、この回避方法によって正しい RL Language が生成されません。これは、`fact class` 文が適切な順序で生成されないためです。この場合、回避方法はありません。

RL Language では、一部の継承階層、特に多重継承を必要とする継承階層は許可されません。たとえば、`Interface2` が `Interface1` を拡張したもので、`Class1` が両方のインタフェースを直接実装する場合は、単一継承ツリーを判別できません。`Class1` が `Interface2` のみを実装する場合は、単一継承ツリーを判別でき、クラスを RL Language で使用できます。

関連項目： 3-9 ページ「データ・モデル内の Java オブジェクトの表示」

索引

A

assertXPath 関数，4-27
Author
　Rule Author を参照

E

EmptyFileRepository，B-6

F

FORM_ADVANCED
　Rules SDK，C-3
FORM_LITERAL
　Rules SDK，C-3

J

Java クラス
　データ・モデルへのインポート，2-11
Java ファクト
　プロパティ変更リスナーを使用，C-15
Java ファクト・タイプ，1-8，2-9
JAXB 生成クラス，4-7
JBoss
　デプロイ，C-8
JDK 1.4.2 クラス
　インポート，D-2
JDK 1.4.2 クラスのインポート，D-2
JSR-94
　RL Language テキストとともに使用，5-4
　Rule Author ルールとともに使用，5-2
　URL とともに使用，5-6
　拡張機能，5-8
　ルール実行セット，5-2

M

mod_oradav モジュール，B-2

N

NULL 値
　ルール式に含める，C-3

O

Oracle Business Rules
　RL Language，1-5
　高可用性，B-7，C-17
　必要な JAR ファイル，C-3
Oracle Business Rules に必要な JAR ファイル，C-3
Oracle ウォレット
　ウォレットの場所の設定，B-4
Oracle ウォレットの場所の設定，B-4

R

Rete アルゴリズム，1-3
RL Language
　構文チェック，3-11
　生成，3-11
　チェック，3-11
　表示，3-11
RL Language のクロス積，C-13
RL 構文のチェック
　Java クラスによる，3-11
　XML スキーマによる，3-11
「RL」タブ，3-11
RL の生成，3-11
Rule Author
　概要，1-4
　起動，2-3
　起動方法，2-3，4-2
　セッション・タイムアウト，A-3
　テスト・ルールセットの使用，3-16
　ホーム・ページ，2-4
　ポップアップ遮断機能，2-17
　ルール，1-6
　「ログイン」ページ，2-3
Rule Author の起動，2-3，4-2
RuleSession
　同期化，C-10
　並行性，C-10

S

SCRATCH ディクショナリ・バージョン，A-3
SDK
　RL の生成，2-27
　概要，1-5，6-1
　クラス，6-2
　データ・モデルの使用，6-4

パターン，6-9
ルール，6-7
ルール・セッションの実行，2-27
ルールセット，6-7
SDK でのファクトのアサート，2-28
single sign-on のタイムアウト，C-16

U

URL
ルールセットまたはルールへのアクセス，C-14

W

WebDAV リポジトリ，1-5
アクセスの確立，6-3
初期化パラメータ，6-2
セキュリティの設定方法，B-4
接続方法，B-3
設定方法，B-2
リポジトリ・タイプ・キー，6-2
WebLogic
デプロイ，C-7
web.xml ファイル
セッション・タイムアウト，A-3

X

XML ファクト
JAXB アンマーシャリング，4-25
JAXB クラス・ディレクトリ，4-8
JAXB 生成クラス，4-7
SDK でのインポート，4-25
「XML スキーマ」フィールド，4-8
アサート，4-27
スキーマのインポート，4-10
「ターゲット・パッケージ名」フィールド，4-8
タイプ，1-8
追加，4-7
XML 文書
アンマーシャリング，4-25

あ

アサート
XPath，4-27
アドバンスト・テスト式
Rule Author
アドバンスト・テスト式，3-13
アドバンスト・テスト式オプション，3-12

い

インポート
Java クラス，2-11

お

オブジェクト・チェーン
「拡張」ボックス，3-11
オブジェクトの参照可能性，3-10
オプション
アドバンスト・テスト式，3-12
別名の使用，3-12
ロギング，3-12

か

カスタマイズ・ルール，2-19，2-24，3-3

く

クラスパス
追加，2-9
クロス積，C-13

け

結果
グローバル変数の使用，3-20
コンテナ・オブジェクトの使用，3-21
判断オブジェクトの使用，3-22
言語の変更
ブラウザの使用，D-4

こ

高可用性，B-7，C-17

さ

削除
ルールセット，2-15
参照可能フィールド
メソッドの参照可能性，3-10

し

式
引用符付き，C-3
式内の引用符，C-3
実行関数，2-28
順連鎖化システム，1-4
初期化パラメータ
WebDAV リポジトリ，6-2
ファイル・リポジトリ，6-2

す

推論サイクル，1-4

せ

制約
　正規表現，3-3
　定義，1-9，3-3
　範囲，3-3
　列挙，3-3
セッション・タイムアウト，A-3

た

タイムアウト
　SCRATCH バージョン，A-3
　single sign-on，C-16

て

定義
　Java ファクト，3-9
　RL 関数，3-7
　RL ファクト，3-5
　XML ファクト，4-7，4-11
　制約，3-3
　変数，3-2
ディクショナリ
　SCRATCH バージョン，A-3
　インポート，3-14
　エクスポート，3-14
　削除，3-14
　作成，2-5，4-3
　説明，1-7
　「ディクショナリ・ディレクトリ」フィールド，2-7，4-5
　「ディクショナリ名」フィールド，2-7，4-5
　ネーミング，A-2
　保存，2-8，4-6
　ロード，2-26，4-26，6-3
ディクショナリのロード，6-3
データ・モデル
　生成，3-11
　定義，1-3，2-9
　定義 XML，4-7
テスト・ルールセット機能，3-16
テンポラリ・ファイル，A-3

と

同期化，C-10
トラブルシューティング，D-1

ね

ネーミング規則
　Rule Author，A-2
　XML スキーマのターゲット・パッケージ名，A-2
　ディクショナリ，A-2
　バージョン，A-2
　別名，A-2
　ルールセット，A-2

は

バージョンのネーミング，2-8，A-2
「パターン定義」ページ
　ポップアップ遮断機能，2-17
パフォーマンス
　ランタイム，C-12

ひ

ビジネス用語
　データ・モデルでの定義，2-13
ビジネス・ルール
　定義，1-2

ふ

ファイル・リポジトリ，1-5
　アクセスの確立，6-3
　コンテンツの更新，B-7
　作成，2-5，4-3
　初期化パラメータ，6-2
　テンポラリ・ファイル，B-7
　リポジトリ・タイプ・キー，6-2
ファイル・リポジトリの作成，B-6
ファクト・タイプ
　Java，2-9
　RL ファクト，3-5
　XML，1-8
ファクト変数へのアクセス，D-2
プロパティ・オブジェクト・チェーン，3-11
プロパティの参照可能性
　オブジェクト，3-10
プロパティ変更リスナー
　Java ファクトで使用，C-15

へ

並行性，C-10
別名
　ネーミング，A-2
別名の使用オプション，3-12

ほ

ポップアップ遮断機能
　使用不可，2-17

め

メソッド・オブジェクト・チェーン，3-11

よ

よくある質問，C-1

ら

ランタイム・パフォーマンス，C-12

り

リポジトリ
  emptyFileRepository，B-6
  WebDAV，1-5
  接続，2-5，4-3
  説明，1-7
  バックアップとリカバリ，B-7
  ファイル，1-5
  ファイル・リポジトリの作成，B-6
リポジトリ・タイプ・キー
  WebDAV リポジトリ，6-2
  ファイル・リポジトリ，6-2
リポジトリの高可用性，B-7
リポジトリへの接続，2-5，4-3

る

ルール
  SDK，6-1
  SDK のインポート，2-25
  SDK の概要，1-5
  URL アクセス，C-14
  アクションの追加，2-21
  エンジン，1-6
  カスタマイズ，2-19，2-24，3-3
  起動，1-4
  式
    NULL 値を含める，C-3
  順連鎖化，1-4
  定義，2-16
  データドリブン，1-4
  「名前」フィールド，2-16
  パターンに対するテストの定義，2-19
  パターンの追加，2-17
  「優先度」フィールド，2-16
  ルール・アクション，1-6
  rule conditions，1-6
ルール・セッション
  実行，2-27
  実行関数の使用，2-28
  ファクトのアサート，2-28
ルールセット
  URL アクセス，C-14
  削除，2-15
  定義，2-15
  ネーミング，A-2
ルール対応 Java アプリケーション，1-9，2-25

ろ

ロギング・オプション，3-12