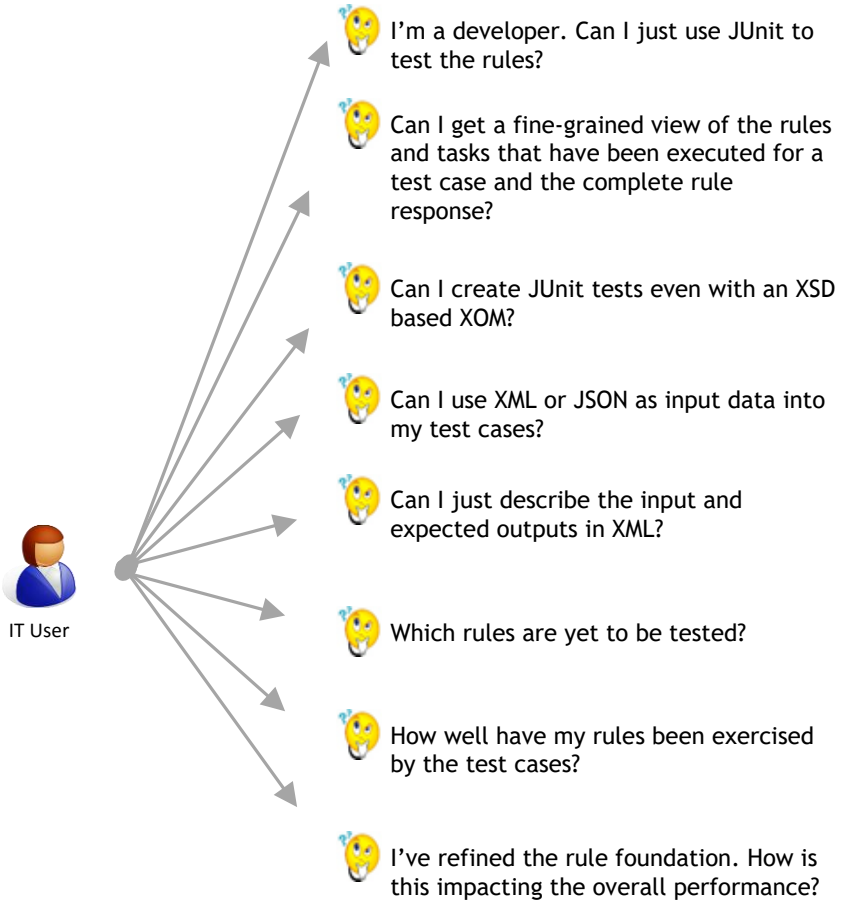
RULE TESTING WITH R_UNIT

INTRODUCTION

Decision Validation Services (DVS) is supplied by IBM as an Operational Decision Manager (ODM) module that enables ruleset testing for developers, business users, and QA engineers. Testing using DVS typically involves creating scenarios (in Excel) that represent real or fictitious use cases to validate the behavior of your rules. However, your organization may have IT-centric development practices and you may prefer to describe your tests in Java or XML or JSON, instead of Excel spreadsheets; perhaps to align with your continuous integration processes. Even if business is eventually responsible for rule validation, as a developer you may want to use your development tools to validate the rule foundation that you've built.

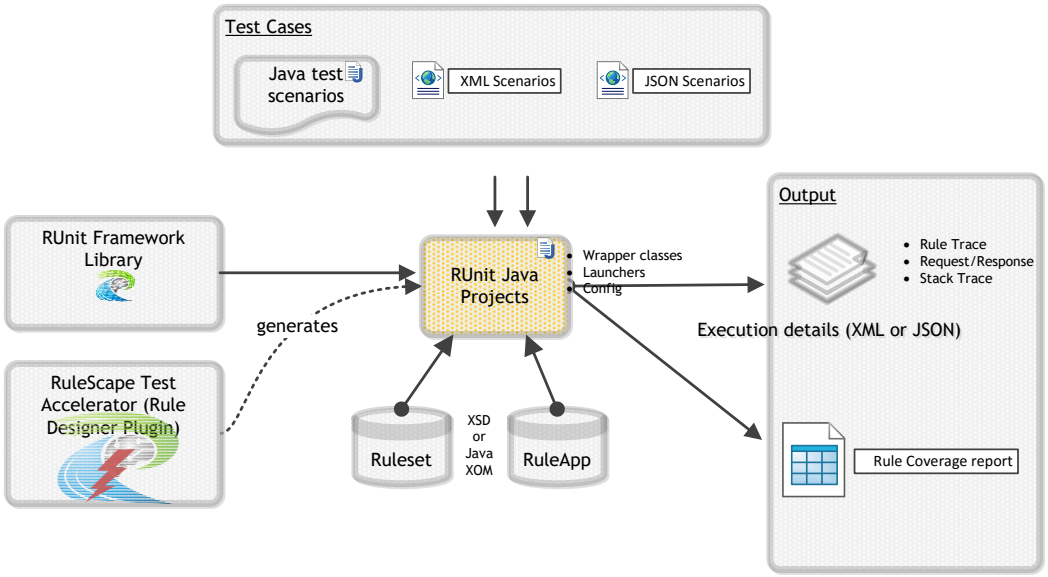
RuleScape R_Unit is the tool you need as a developer if you encounter any of these questions:



RULESCAPE R_UNIT

RuleScape R_Unit is a JUnit-based framework that allows you to easily create test cases in Java, XML or JSON. In addition, it generates detailed logs and reports that contain insightful execution details that help you devise better test cases, identify areas which need more testing, and determine the performance impact of any tuning you've performed.

R_Unit is packaged as an Eclipse plug-in Test Accelerator which generates the Java projects for you to execute the framework.



When test cases are executed with R_Unit, it automatically generates a **Coverage Report** which provides rich details relating to how well your tests cover your rules. This helps you to decide which rules now need to be your focus of testing. Besides the coverage report, R_Unit can be configured to log the inputs, outputs, rule traces and exception stack traces for your test run. These can serve as a baseline for future regression tests.

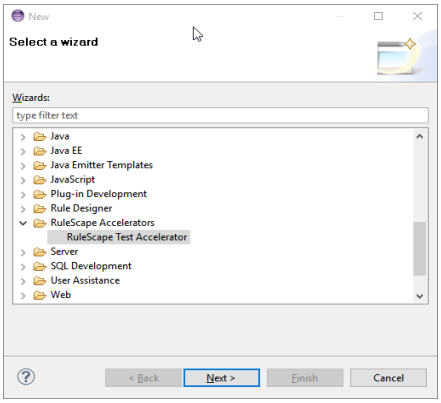
R_Unit recognizes that there are situations where some of the rules may not be in scope for testing purposes; for instance, rules that will only become effective next year. As part of its configuration, R_Unit allows you to identify out-of-scope rules that you want to ignore for test coverage reporting. R_Unit brings this into consideration when calculating the *effective* coverage.

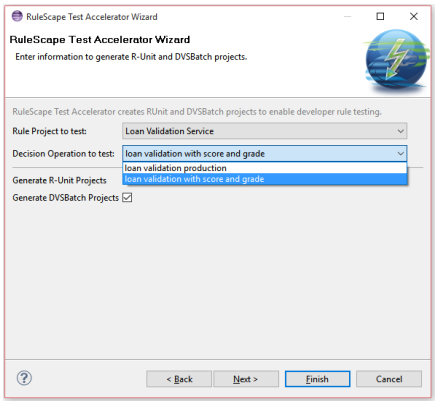
R_Unit makes it very easy for you to start testing any ruleset you've developed. The straight-forward steps to using R_Unit tool are:

1. Generate pre-configured R_Unit projects using **Test Accelerator Plug-in.**
2. Add additional configuration parameters if necessary.
3. Tweak the generated sample test cases by creating meaningful test cases in Java, XML or JSON. Specify the expected results in XML or through JUnit Assert statements.
4. Execute the tool using one of the provided launchers and gain insights through the **Excel Coverage Report** and other execution logs.

TEST ACCELERATOR PLUG-IN

Test Accelerator, an Eclipse plug-in that extends ODM Rule Designer, provides a simple wizard that generates preconfigured R_Unit projects based on the rule project or decision operation you've selected to test.





The Test Accelerator generates all the scaffolding necessary for you to execute R_Unit, which includes framework libraries, preconfigured Java projects (with samples on how to invoke R_Unit API) and launchers to execute your R_Unit test cases.

R_UNIT PROJECT CONFIGURATION AND EXECUTION

The Test Accelerator generates pre-configured R_Unit projects. You can easily reconfigure test execution using the supplied configuration file. Common configuration changes include:

- Logo to be used in reports
- Identification of out-of-scope rules. These are rules that are present in the ruleset, but are not in scope for current testing and should be excluded for coverage purposes.
- Location of folders for Regression data tests, Triage data tests and reports.

Several launchers, for different test scopes, are provided in the R_Unit project. Once R_Unit has been configured and the test cases defined, execution of the tests is simply a matter of running these launchers. The test execution generates log files and a coverage report in the *reports* folder.

R_UNIT API

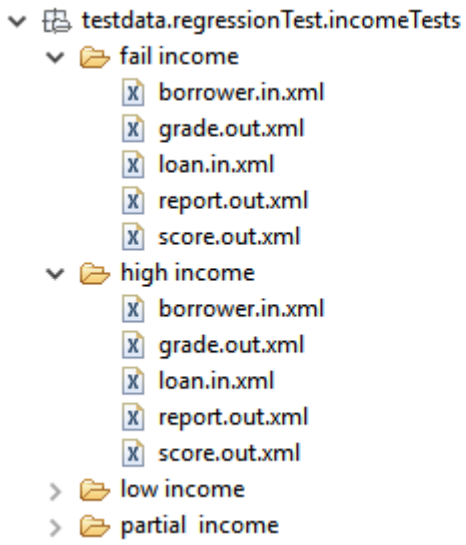
R_Unit framework provides a simple API that hides all the complexity of creating rule engines, loading the ruleset or RuleApp, executing rules, and gathering detailed traces. It also provides API to help construct rule requests from XML and JSON, and compare with expected results in XML.

The API makes it easy to create test cases that execute rules. A sample, with highlighted R_Unit API calls in yellow and generated wrapper elements in gray is shown below:

```
public void testScenario1Json() {

   // create request from json
   Borrower borrower1 =  TestHelper.buildObject("input_borrower1.json",
Borrower.class, jsonReadXstream);
   LoanRequest loan1 =  TestHelper.buildObject("input_loan1.json",
LoanRequest.class, jsonReadXstream);


   // execute rules
   LoanValidationServiceRuleExecutionResult ruleResp = executeRuleset(borrower1,
loan1);

   Report reportValue = ruleResp.getReport();
   Integer scoreValue = ruleResp.getScore();
   String gradeValue = ruleResp.getGrade();

   // run tests
   assertNotNull(reportValue);
   assertNotNull(scoreValue);
   assertNotNull(gradeValue);

   assertTrue(hasRuleFired("eligibility.checkIncome"));
   assertFalse(reportValue.isApproved());
 }
```

R_UNIT REGRESSION TESTS

You can define entire scenarios, including the expected results, in XML format. Each of the ruleset parameters, both input and output, is defined in a separate XML file. These can be placed the regression test data folder and executed automatically by the framework.



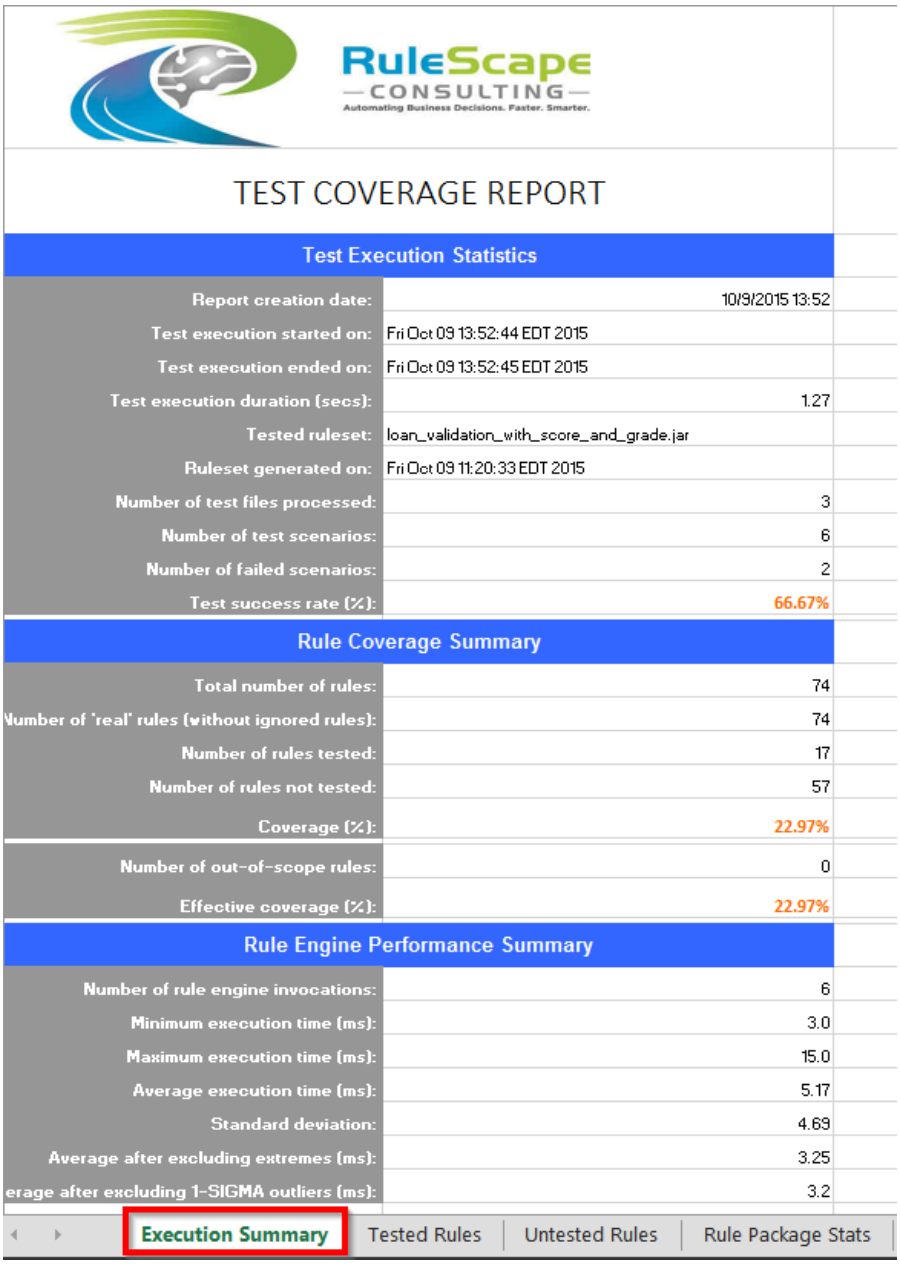
COVERAGE REPORT

R_Unit generates an Excel Coverage Report that provides you with deep insights relating to the test execution. This report has four worksheets:

1. Execution Summary
2. Tested Rules
3. Untested Rules
4. Rule Package Statistics

EXECUTION SUMMARY

The Execution Summary report tells you at a glance how well your tests ran. You can gauge the validity of your ruleset through the test success rate, and gauge performance through the Rule Engine Performance Summary. In addition, the Rule Coverage Summary gives you a snapshot of how many rules were not tested.



RuleScape CONSULTING
Automating Business Decisions. Faster. Smarter.

TEST COVERAGE REPORT

Test Execution Statistics	
Report creation date:	10/9/2015 13:52
Test execution started on:	Fri Oct 09 13:52:44 EDT 2015
Test execution ended on:	Fri Oct 09 13:52:45 EDT 2015
Test execution duration (secs):	1.27
Tested ruleset:	loan_validation_with_score_and_grade.jar
Ruleset generated on:	Fri Oct 09 11:20:33 EDT 2015
Number of test files processed:	3
Number of test scenarios:	6
Number of failed scenarios:	2
Test success rate (%):	66.67%

Rule Coverage Summary	
Total number of rules:	74
Number of 'real' rules (without ignored rules):	74
Number of rules tested:	17
Number of rules not tested:	57
Coverage (%):	22.97%
Number of out-of-scope rules:	0
Effective coverage (%):	22.97%

Rule Engine Performance Summary	
Number of rule engine invocations:	6
Minimum execution time (ms):	3.0
Maximum execution time (ms):	15.0
Average execution time (ms):	5.17
Standard deviation:	4.69
Average after excluding extremes (ms):	3.25
erage after excluding 1-SIGMA outliers (ms):	3.2

Execution Summary | Tested Rules | Untested Rules | Rule Package Stats

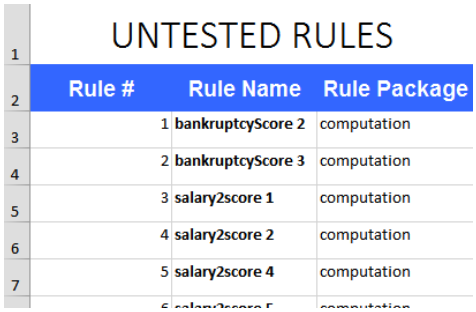
TESTED RULES

The Tested Rules worksheet depicts how well a rule has been tested and provides details on which scenarios tested each of the rules. This information is valuable in determining which test conditions a particular rule has been tested under. This information can guide you in devising more significant tests for a rule.

Rule #	Rule Name	Rule Package	Total Executions	Executed in Scenarios
1	initialCorporateScore	computation	6	demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.high income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.low income
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1Json
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1XML
				demo.ruletests.scenarios.LoanValidationServiceTest.testScenario1
2	neverBankruptcy	computation	6	demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.high income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.low income
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1Json
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1XML
				demo.ruletests.scenarios.LoanValidationServiceTest.testScenario1
3	rate 3	computation	6	demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.high income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.low income
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1Json
				demo.ruletests.scenarios.LoanValidationServiceDataTest.testScenario1XML
				demo.ruletests.scenarios.LoanValidationServiceTest.testScenario1
4	repayment	computation	6	demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income
				demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.high income

UNTESTED RULES

Untested Rules worksheet lists out rules that have not been tested by any of the scenarios in this test run. When your entire set of DVS scenarios is run as a regression test, this list reveals which rules are yet to be tested and serves as a guide for helping you determine the area of test focus.



UNTESTED RULES

Rule #	Rule Name	Rule Package
1	bankruptcyScore 2	computation
2	bankruptcyScore 3	computation
3	salary2score 1	computation
4	salary2score 2	computation
5	salary2score 4	computation

RULE PACKAGE STATS

The Rules Package Stats worksheet displays test coverage details at the rule package level. For each rule package, the calculated effective coverage rate is helpful in determining which package needs attention from testers.

Rule Package	Rules Tested	Total Rules	Rules Out of Scope	Test Coverage	Effective Coverage
ROOT:	17	74	0	22.97%	22.97%
computation:	6	39	0	15.38%	15.38%
eligibility:	3	15	0	20.00%	20.00%
insurance:	3	14	0	21.43%	21.43%
validation:	5	6	0	83.33%	83.33%
borrower:	5	5	0	100.00%	100.00%
loan:	0	1	0	0.00%	0.00%

EXECUTION LOGS

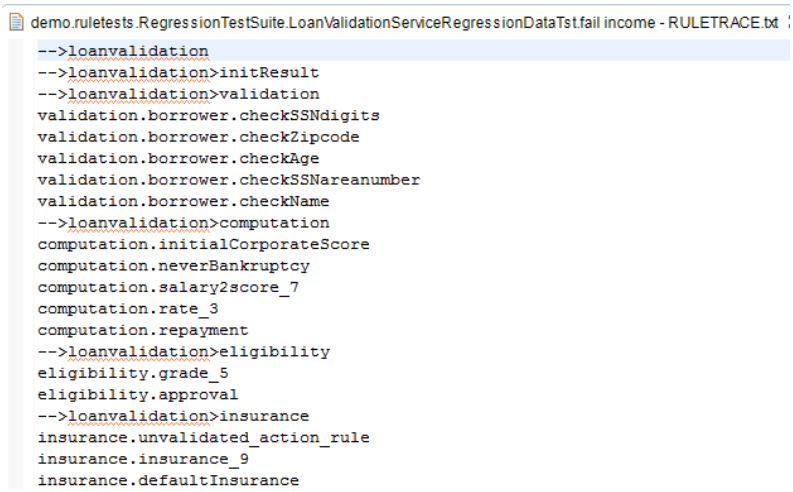
R_Unit generates text files containing the rule trace and execution details for any failed test case, but can be configured to do so for all test cases.

The execution details of each scenario is placed in a separate, easily identifiable text file. It not only contains the test results identifying any discrepancies between expected and actual outcomes, but also logs the request data and the entire observed response in a human-readable format.

demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income - FAILED - DETAILS.txt

```
------------------------------------------------------------------
  TEST RESULTS
------------------------------------------------------------------
Status: FAILED
Duration (in secs): 0.038
2 differences detected
text value at /loan.Report[1]/messages[1]/string[2]/text()[1] :
    EXPECTED VALUE=Congratulations! Your loan has NOT been approved , OBSERVED VALUE= Congratulations! Your loan has been approved
text value at /loan.Report[1]/yearlyInterestRate[1]/text()[1] :
    EXPECTED VALUE=0.065 , OBSERVED VALUE= 0.055

------------------------------------------------------------------
  OBSERVED RESPONSE
------------------------------------------------------------------

**ilog.rules.firedRulesCount:
<int>15</int>

**report:
<loan.Report>
  <borrower>
    <firstName>Smith</firstName>
    <lastName>John</lastName>
    <birth>
      <time>-628455600000</time>
      <timezone>America/New_York</timezone>
    </birth>
    <SSN>
      <areaNumber>123</areaNumber>
      <groupCode>12</groupCode>
      <serialNumber>1234</serialNumber>
      <outer-class reference="../.."/>
    </SSN>
    <yearlyIncome>200000</yearlyIncome>
    <zipCode>12345</zipCode>
    <creditScore>200</creditScore>
  </borrower>
  <loan>
    <numberOfMonthlyPayments>48</numberOfMonthlyPayments>
    <startDate>2015-09-05 13:11:26.763 EDT</startDate>
    <amount>100000</amount>
    <loanToValue>1.2</loanToValue>
  </loan>
  <validData>true</validData>
  <insuranceRequired>true</insuranceRequired>
  <insuranceRate>0.02</insuranceRate>
  <approved>true</approved>
  <messages>
    <string>Average risk loan</string>
    <string>Congratulations! Your loan has been approved</string>
  </messages>
  <yearlyInterestRate>0.055</yearlyInterestRate>
```

Rule trace lists all the rule tasks and rules invoked during execution of the test case. This information is valuable as a debugging tool. In addition, it provides a baseline to compare and identify which rule is failing to run for the same test scenario on a future date.


demo.ruletests.RegressionTestSuite.LoanValidationServiceRegressionDataTst.fail income - RULETRACE.txt

```
-->loanvalidation
-->loanvalidation>initResult
-->loanvalidation>validation
validation.borrower.checkSSNdigits
validation.borrower.checkZipcode
validation.borrower.checkAge
validation.borrower.checkSSNareanumber
validation.borrower.checkName
-->loanvalidation>computation
computation.initialCorporateScore
computation.neverBankruptcy
computation.salary2score_7
computation.rate_3
computation.repayment
-->loanvalidation>eligibility
eligibility.grade_5
eligibility.approval
-->loanvalidation>insurance
insurance.unvalidated_action_rule
insurance.insurance_9
insurance.defaultInsurance
```

RULESCAPE R_UNIT BENEFITS

R_Unit improves the productivity and efficiency of rule developers, especially when the rule validation is primarily IT driven. It greatly enhances the robustness of the rule foundation created by IT developers.

- ✓ The Test Accelerator provides a fast and easy mechanism for Java developers to get started with rule testing.
- ✓ Allows developers to pick the data format of their choice for testing.
- ✓ Java based test cases are automatically updated by the IDE in case of model refactoring.
- ✓ The framework is simple to use and easily configurable.
- ✓ Provides managers a means to track test progress, gauge robustness of the ruleset and identify any areas of weakness.
- ✓ Guides users in identifying rules that should be the focus of their testing.
- ✓ Enables developers to assess performance impact of any tuning they've performed.