



VOYAGER *Wizard*[™] Ruleset Builder User's Guide

Version 1.0 for Voyager 8.0-RC3

Table of Contents

Introduction	4
Overview.....	4
Preface.....	4
Ruleset Builder Requirements	4
JBoss Rules information	4
Contacting Technical Support	5
Inference Rules Overview.....	5
Basic Rule concepts	5
Rules Example	6
Voyager Rules.....	6
Rulesets.....	6
Rule.....	7
Voyager Ruleset Builder.....	9
Ruleset Builder Top Component	9
Ruleset Editor.....	10

<This page intentionally blank>

Introduction

Overview

Voyager™ provides the ability to run inference engine rules in a mobile agent and report the results back to the caller. To aid in writing these rules we provide the VOYAGER *Wizard*™ Ruleset Builder. Using the Ruleset Builder you can write Goal oriented rules.

Preface

The purpose of this manual is to provide an introduction to basic Ruleset Builder features as they pertain to Voyager. This is not meant to be a treatise on writing inference rules. This guide assumes a basic knowledge of Java and distributed computing concepts. Voyager uses the JBoss Rules engine to execute inference rules.

This preface covers the following topics:

- Ruleset Builder Requirements
- JBoss rules information
- Contacting technical support

Ruleset Builder Requirements

Before attempting to use the Ruleset Builder features of Voyager.

- A Java Development Kit (JDK) installed on your computer. Voyager requires JDK 1.4.2 or later. You can download the latest release of JDK from www.javasoft.com at no charge.
- A working version of Voyager installed.

JBoss Rules information

Information about JBoss Rules syntax can be found in the following places:

Copyright © 2006 – 2010 Recursion Software, Inc.
All Rights Reserved

- Main site: <http://labs.jboss.com/portal/jbosrules/index.html>
- Documentation: <http://labs.jboss.com/portal/jbosrules/docs/index.html>

See the documentation for information on the syntax of rule conditions (left hand side of a rule) and consequence. (right hand side of a rule) The Ruleset Builder provides a framework for writing inference rules, but the syntax of the rule parts must still be compiled and executed by the JBoss Rules engine.

Contacting Technical Support

Recursion Software welcomes your problem reports, and appreciates all comments and suggestions for improving Voyager. Please send all feedback to the Recursion Software Technical Support department.

Technical support for Voyager is available via the web, email, and phone. The Recursion Software Support website, at <http://support.recursionsw.com>, provides access to searchable FAQs (Frequently Asked Questions), technical articles, and other information. It also provides access to the Recursion Software Technical Support incident tracking system, where you can submit support issues and track them. You can also contact Technical Support by sending email to psupport@recursionsw.com or by calling (972) 731-8800.

Inference Rules Overview

Basic Rule concepts

An inference engine is a tool that uses a set of if/then/else rules to analyse and manipulate a set of data objects. The rules are generally referred to as a Ruleset, and the objects are the Working Memory. A rule consists of two parts, the right hand side, or RHS, which is a set of conditions that matches on objects in Working Memory. The left hand side of a rule (LHS) is a list of actions that act on the Working Memory. The LHS of a rule can retract an object from Working Memory, modify an object in Working Memory, or add a new object to Working Memory. All of these actions can change which rules match their right hand side conditions, thus changing which left hand side actions are subsequently taken. This process of matching condition, taking actions, then re-evaluating the conditions that apply is called Forward Chaining.

Rules Example

A classic example in inference engine literature is the “Monkeys and Bananas” scenario. In this example, the Working Memory consists of facts about a monkey and it’s environment: furniture in a room, the monkey’s position in the room, and a bunch of bananas hanging from the ceiling. The rules consist of conditions to be met and actions to be taken for the monkey to get the bananas, which are out of it’s reach on the ceiling. Some sample rules are the following:

If <Monkey: not holding bananas> then <Assert Goal: Get Bananas>

If <Goal: Get Chair> and <Monkey: not at chair> then <Modify Monkey: Move to chair>

If <Goal: Get Bananas> and <Monkey: has no bananas and has no chair> then <Assert Goal: Get Chair>

For brevity’s sake these rules are simplistic. They could be arbitrarily complex. For instance, they could direct the monkey to find the bananas by looking up while walking to different positions in the room, searching for the chair in the same manner, or stacking different objects until a tall enough pile is made to reach the bananas. The important part to remember is that the rules are not executed in any particular order. A rule will fire only when it’s conditions are met, and not in any particular order.

Voyager Rules

Rules built by the Voyager Ruleset Builder follow the convention that each rule is meant to help accomplish a goal. Goals are used to structure rule processing so the rule writer has more control over the order of rule execution. Voyager rules must specify a goal to be addressed, whether the rule accomplishes the goal, and the next goal to accomplish, if any.

Rulesets

A Ruleset is the top level data structure in the Voyager rule system. It contains four major parts:

- Goals
- Import classes
- Global variables
- Rules

The list of Goals contains all Goals used for Ruleset processing. Each rule must have a Goal, and they can only have a Goal taken from this list. Goals should have simple, well understood names to make execution order understandable. Most Rulesets should have at least two Goals named “BEGIN_PROCESSING” and “END_PROCESSING”. This is not a requirement, but a good convention to follow.

The set of import classes consists of the names of all classes used in the execution of the rules. This includes all objects in Working Memory, all global variables, and all classes used in the execution of rule consequences. Basically, if you use a class in the rules, you must list it here, otherwise the Ruleset will not execute.

The set of global variables describe objects the rules use when they execute. Globals can only be used in the consequence (LHS) of a rule, not in it’s conditions. (RHS) When a Voyager agent executes a Ruleset, it sends back the contents of the Globals when finished. A Global consists of two parts: name and type. The type is a Java class (listed in the Imports list above) that must implement java.io.Serializable.

The most important part of a Ruleset is it’s list of rules. Components of a rule are described below.

Rule

A rule consists of the following parts:

- A Goal
- A Priority value
- Conditions
- A Consequence, consisting of
 - Action text
 - A true/false value for whether the rule’s Goal was accomplished
 - The next Goal to accomplish, if any

As explained above, Goals allow the programmer to organize rules into execution groups. Each rule must have one and only one Goal.

A rule’s priority is also a form of execution organization. Rules with a higher priority value execute before rules with a lower priority. Use priority to influence execution order within a Goal.

Conditions are logical tests against objects in Working Memory. Conditions must conform to JBoss Rules syntax, which can be found in the JBoss Rules User Guide in the “Conditional Elements” section:

<http://labs.jboss.com/portal/jbossrules/docs/index.html>

```
Person( age >= 35, sex = "M" )
```

Conditions can also instantiate variables to be used in the rule Consequence.

```
bugGuy : Person(sex = "M", heightInInches > 72)
hisWife : Person(sex = "F", marriedTo == bigGuy)
```

Conditions interact by instantiating variables and using them in logical conditions. For instance, to match pets of the same color, these Conditions would suffice:

```
pet1 : Pet(colorHolder : color)
pet2 : Pet(color == colorHolder)
eval(pet1 != pet2)
```

The rule Consequence is where you program actions to be taken by the rule once it matches Working Memory objects. The Consequence text can be any valid java code. In addition, there are three methods that can be used to manipulate the Working Memory objects matched in the Conditions or add new ones: assert, modify, and retract. Assert adds new objects to Working Memory. Modify informs the inference engine that an object has been modified. Retract removes an object from Working Memory. For instance, the following Consequence calls the setImmunized() method on the two matched pets and tells the inference engine that they’ve been modified:

```
pet1.setImmunized(true);
pet2.setImmunized(true);
modify(pet1);
modify(pet2);
```

Voyager rule Consequences also must indicate whether their rule’s Goal has been accomplished, and indicate what the next Goal is in the sequence. These two operations will retract the old Goal or assert the new Goal, respectively.

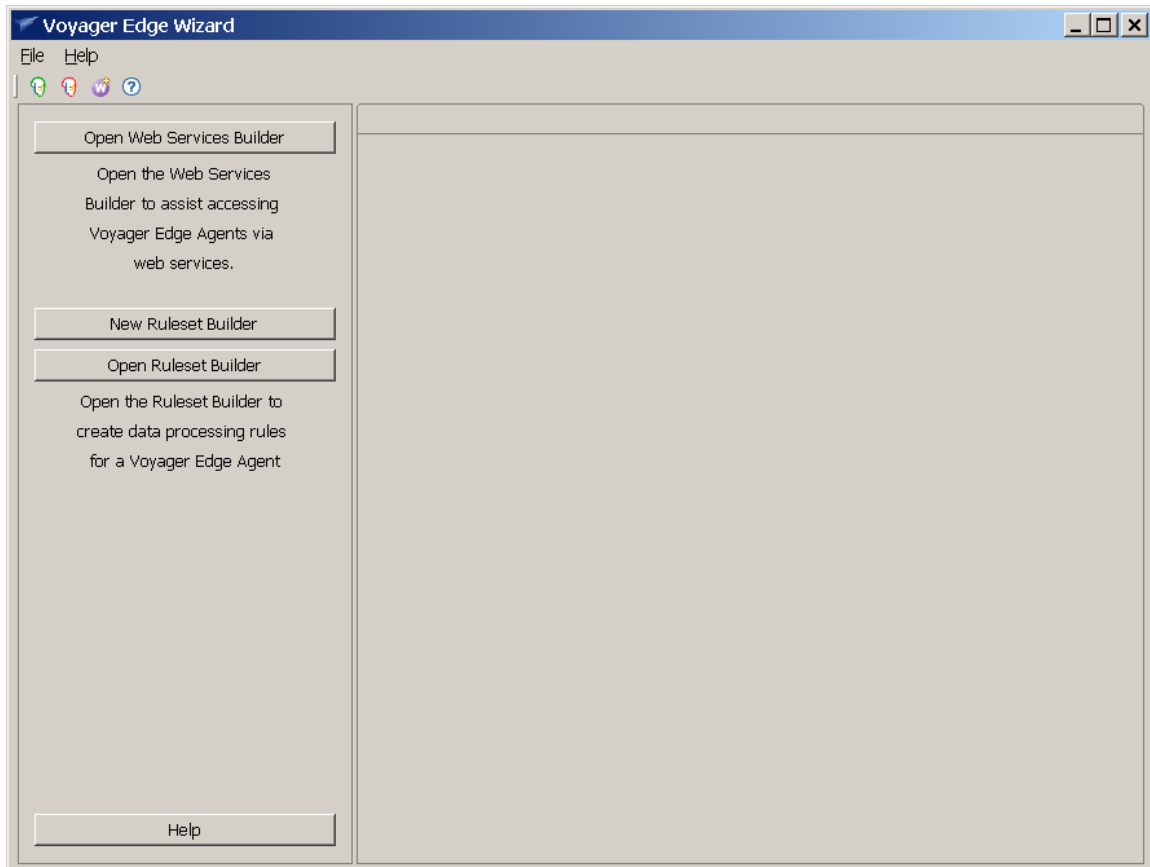
Voyager Ruleset Builder

The Voyager Ruleset Builder can be found in the %VOYAGER_HOME%/bin/wizard/eclipse directory. To start the builder execute VoyagerWizard.exe. Below is a description of the graphical components of the Voyager rule system.

Ruleset Builder Top Component

From the main application window, select the File menu. Two menu options should be available for rules: Create New Ruleset and Open Existing Ruleset. Options for Webservices are not covered in this document.

- New: Create a new Ruleset, with default Goals, Imports, and Globals
- Open: Open an existing Voyager Ruleset file (.vrs)

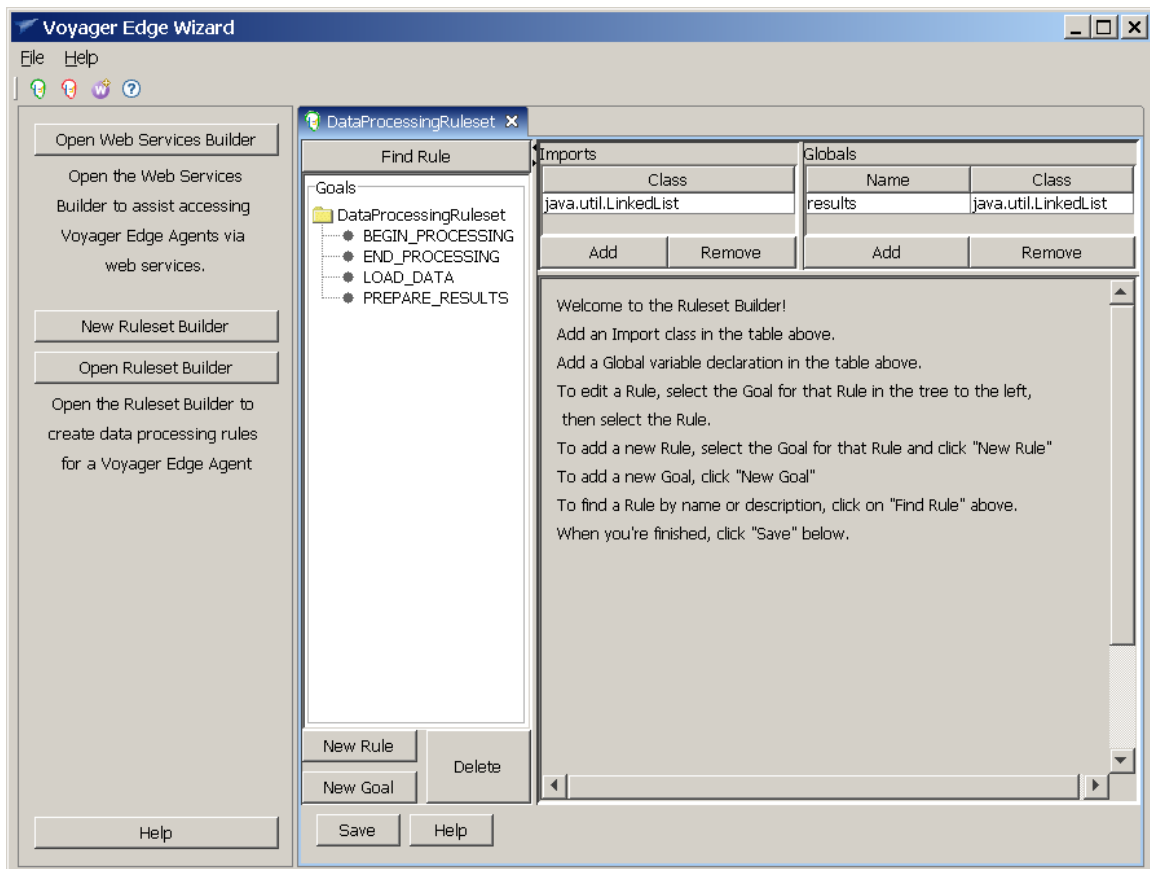


Copyright © 2006 – 2010 Recursion Software, Inc.
All Rights Reserved

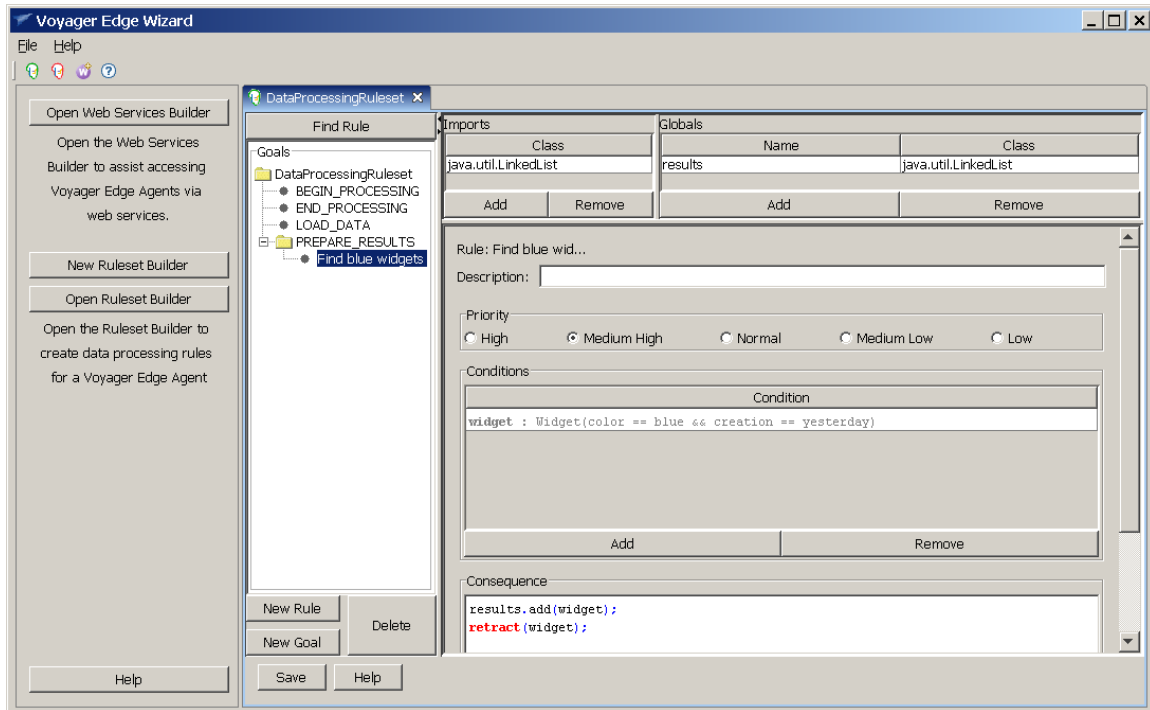
Ruleset Editor

After opening or creating a new Ruleset, the Ruleset Editor will appear. To the left is a tree component listing all of the rules, organized by Goal. On the top are tables holding the Imports and Globals. If you select a rule on the left, the Rule Editor will appear in the center, populated with the settings of that rule.

This is the Ruleset Editor after a new Ruleset has been opened.



This is the Ruleset Editor after a new rule has been created:



New Rule Wizard

A new rule can be created by clicking on the "New Rule" button on the lower left, beneath the rule tree. Step through the wizard and fill in the necessary data to create the rule. The only required field is the "name" field on the first panel. Note: **A rule name must be unique within its Ruleset.**

New Rule Wizard

Name New Rule

Rule Name

Goal

Description

Please enter basic rule information.
You cannot proceed until you enter a Rule name.

Help < Back Next > Finish Cancel



