JBoss ESB 4.2 GA

Content Based Routing

JBESB-CBR-8/29/07

Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

Software Version

JBoss ESB 4.2 GA

Restricted Rights Legend

Contents

Table of Contents

About This Guide

What This Guide Contains

The Content Based Routing contains contain important information on changes to JBoss ESB 4.2 GA since the last release and information on any outstanding issues.

Audience

This guide is most relevant to engineers who are responsible for administering JBoss ESB 4.2 GA installations.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, What is Content-Based Routing:** An overview of why you would want to use CBR.
- **Chapter 2, Content-Based Routing:** this chapter contains information on how to use the content based routing capabilities in JBossESB.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
`Code`	Text that represents programming code.
Function | Function	A path to a function or dialog box within an interface. For example, "Select File | Open." indicates that you should select the Open function from the File menu.
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax: `persistPolicy (Never | OnTimer | OnUpdate | NoMoreOftenThan)`
Note: **Caution**:	A note highlights important supplemental information. A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBoss ESB 4.2 GA documentation set:

1. **JBoss ESB 4.2 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.

2. **JBoss ESB 4.2 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.

3. **JBoss ESB 4.2 GA** *Programmers Guide*: How to use JBossESB.

4. **JBoss ESB 4.2 GA** *Release Notes*: Information on the differences between this release and previous releases.

5. **JBoss ESB 4.2 GA** *Administration Guide*: How to manage the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.2 GA should be directed to our support team.

Chapter 1

What is Content-Based Routing?

Introduction

Typically information with the ESB is conveniently packaged, transferred, and stored in the form of a message. Messages are addressed to Endpoint References (services or clients), that identify the machine/process/object that will ultimately deal with the content of the message. However, what happens if the specified address is no longer valid? For example, the service has failed or been removed from service? It is also possible that the service no longer deals with messages of that particular type; in which case, presumably some other service still handles the original function, but how should the message be handled? What if other services besides the intended recipient are interested in the message's content? What if no destination is specified?

One possible solution to these problems is *content-based routing*. Content-based routing seeks to route messages, not by a specified destination, but by the actual content of the message itself. In a typical application, a message is routed by opening it up and applying a set of rules to its content to determine the parties interested in its content.

The ESB can determine the destination of a given message based on the content of that message, freeing the sending application from having to know anything about where a message is going to end up.

Content-based routing and filtering networks are extremely flexible and very powerful. When built upon established technologies such as MOM (Message Oriented Middleware), JMS (Java Message Services), and XML (Extensible Markup Language) they are also reasonably easy to implement.

Simple example

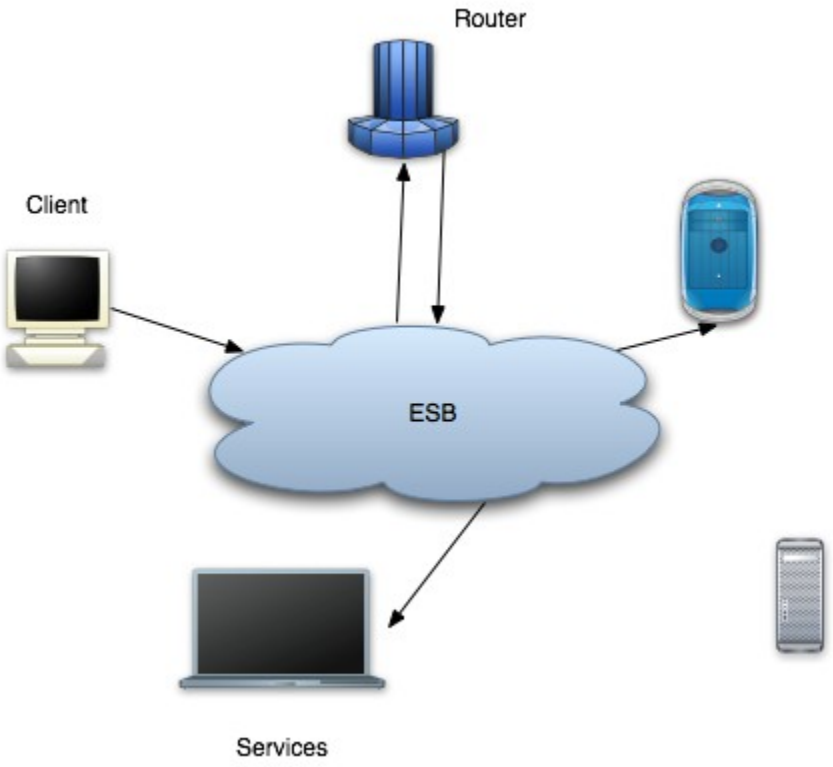
Content-based routing systems are typically built around two types of entities: *routers* (of which there may be only one) and *services* (of which there is usually more than one). Services are the ultimate consumers of messages. How services publish their interest in specific types of messages with the routers is implementation dependent, but some mapping must exist between message type (or some aspect of the message content) and services in order for the router to direct the flow of incoming messages.

Routers, as their name suggests, route messages. They examine the content of the messages they receive, apply rules to that content, and forward the messages as the rules dictate.

In addition to routers and services, some systems may also include *harvesters*, which specialize in finding interesting information, packaging it up as a formatted message

before sending it to a router. Harvesters mine many sources of information including mail transfer agent message stores, news servers, databases and other legacy systems.

The diagram below illustrates a typical CBR architecture using an ESB. At the heart of the system, represented by the cloud, is the ESB. Messages are sent from the client into the ESB, which directs them to the Router. This is then responsible for sending the messages to their ultimate destination (or destinations, as shown in this example).



Chapter 2

Content Based Routing

Introduction

The Content Based Router (CBR) in the JBossESB can be used to route message to the next destination based on the content of the message. By default the ESB will use JBossRules as it's evaluation engine, however this is left configurable. In the jbossesb-properties.xml there is property called 'org.jboss.soa.esb.routing.cbrClass' in the 'messagerouting' section:

```
<properties name="messagerouting">
  <property name="org.jboss.soa.esb.routing.cbrClass"
            value="org.jboss.internal.soa.esb.services.routing.cbr.JBossRulesRo
uter"/>
</properties>
```

which can be used to plug in another evaluation engine.

JBossRules based Router

By default the Content Based router uses JBossRules. Read on to find how to bring up CBR, how to give it a specific rule set, and how to send a message to the CBR.

Create a Content-Based Router

To bring up a Content-based router you need to bring up the CBR-service. Currently this is done by bringing up ContentBasedRouting Service. To do this you need to add an xml-chapter to your deployment-config.xml that looks like the following example which uses JMS as transport protocol:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jbossesb
xmlns="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/etc/schemas/xml/jbossesb
-1.0.xsd"
parameterReloadSecs="10">

        <providers>
          <jms-provider name="localhost"
                      connection-factory="ConnectionFactory"
                      jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
                      jndi-URL="localhost" >

              <jms-bus busid="QueueA">
                  <jms-message-filter
                      dest-type="QUEUE"
                      dest-name="queue/A"
                      selector="service='CBRouting-SerializableOrXml'"
                   />
              </jms-bus>
          </jms-provider>
      </providers>
      <services>
        <service
               category="MessageRouting"
               name="ContentBasedRoutingService"
```

```
                description="CBR Listener">
                <listeners>
                   <jms-listener name="XPathContentBasedRouter"
                                 busidref="QueueA"
                                 maxThreads="1">
                    </jms-listener>
              </listeners>
                <actions>
                     <action class="org.jboss.soa.esb.actions.ContentBasedRouter"
                                                 name="ContentBasedRouter">
                     <property name="ruleSet" value="JBossESBRules.drl"/>
                     <property name="ruleReload" value="true"/>
                     <property name="destinations">
                         <route-to destination-name="xml-destination-nodelivery" service-
                                                  category="category01" service-
name="jbossesbtest1" />
                         <route-to destination-name="serialized-destination-nodelivery"
service-                                         category="category02" service-
name="jbossesbtest2" />
                     </property>
                     <property name="object-paths">
                       <object-path path="body.test1" />
                       <object-path path="body.test2" />
                     </property>
                  </action>
                </actions>
        </service>
   </services>
</jbossesb>
```

Define a service and add a ContentBasedRouter

Cbr specific properties to the (jms-)listener:

1. ruleSet: The set of rules that should be used to evaluate the content. Only 1 ruleSet can be given for each CBR. Just define more router if you need composite CBR routing.

2. ruleLanguage: In JBossRules you can define a custom (business) language. This (optional) field can be used to pass in that custom language to go with the defined ruleSet.

3. RuleReload: the rules will reload 'hot' if refreshed

4. A set of destinations. The destination-name is what will be used in the ruleSet. The service-category and service-name are then used to deliver the message.

Note that you also have to define the following dependency in your deployment.xml file:

```
<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbrules.esb</depends>
</jbossesb-deployment>
```

Create a ruleSet

A rule set can be created using the JBossIDE which includes a plug-in for JbossRules. Figure 1 shows a screen shot the plug-in.

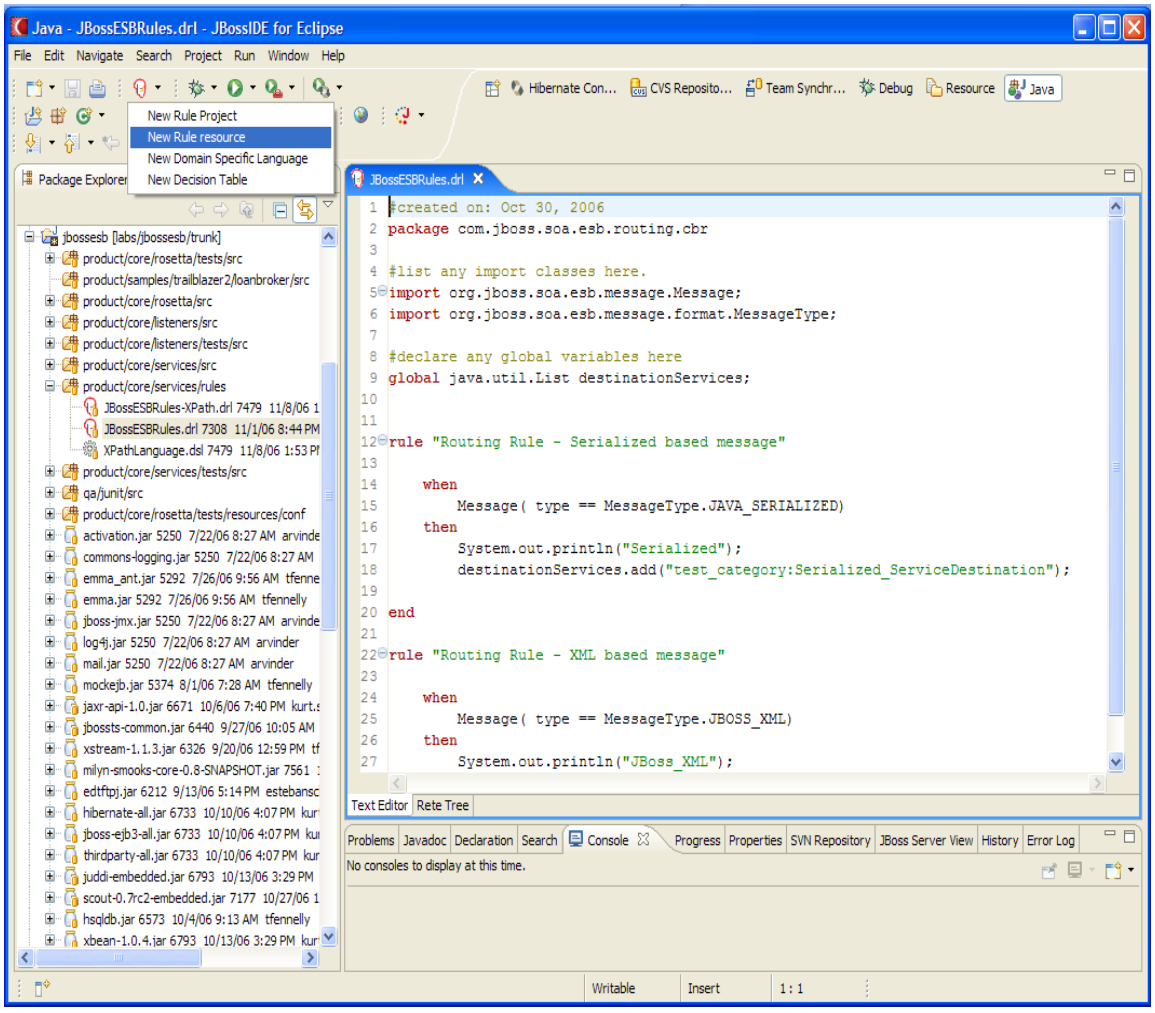


Figure 1. JBoss Rules IDE. Create a new ruleSet.

To add a new rule simple click on "New Rule resource". Make sure your rules import at least

```
import org.jboss.soa.esb.message.Message;
```

The message will be asserted into the working memory of the rules engine. You can the values of parameters on the message in your evaluation. This is demonstrated in the JBossESBRules.drl ruleSet where the rule checks if the type is XML or Serializable. Note that JbossRules treats objects as shallow objects to achieve highly optimized performance, so what if you want to evaluate an object deeper in the object tree? Either you make serialize your message to XML, and you can use XPATH, or you can sepcify 'object-paths', which extracts objects from the message, using an ESB Message Object Path. The path should follow the syntax:

location.objectname.[beanname].[beanname]...

location : one of {body, header, properties, attachment}

objectname: name of the object name, attachments can be named or numbered, so for attachments this can be a number too.

beannames: optionally you traverse a bean graph by specifying bean names; examples :

properties. Order, gets the property object named "Order"

attachment.1, gets the first attachment Object

attachment.FirstAttachment, gets the attachment named 'FirstAttachment'

attachment.1.Order, calls getOrder() on the attached Object.

body.BODY_CONTENT, gets the byte[] of the body.

body.Order1.lineitem, obtains the object named "Order1" from the body of the message. Next it will call getLineitem() on this object. More elements can be added to the query to traverse the bean graph.

Make sure to import the objects into your rule. The Object Mapper cannot traverse collections, so if you need to do that you have to a (Smooks-) transformation on the message first, to unroll the collection.

Secondly you have to make sure to define:

```
global java.util.List destinationServices;
```

The destinationServices List is returned and should contain one or more destination services. The format of the destination service should be:

<service-category>:<service-name>

If you save your rule into the product/core/services/rules package it will be jarred up into the jbossesb-rules.jar. However as long as you your rules on the classpath somewhere JBossRules should find it.

Use the XPath Custom Rule Language

For XML-based messages it is convenient to do XPath based evaluation. For this we ship a custom rule language defined in the XpathLanuage.dsl. To use it you need to reference it in your ruleSet with:

```
expander XPathLanguage.dsl
```

and you need to pass it in the name of this file in the attributes like:

```
ruleSet="JBossESBRules-XPath.drl"
ruleLanguage="XPathLanguage.drl"/>
```

Currently the XPath Language makes sure the message is of the type JBOSS_XML and it defines

1. xpathMatch "<element>": yields true if is an element by this name is matched.

2. xpathEquals “<element>”, “<value>”): yields true if the element is found and it's value equals the value.

You can define your own Custom Rule Language.

Splitter

To make a message go to more then 1 destination (split) simply set up your rules to match more then one destinations. You can also use the Static Router with multiple destinations to achieve this. In either the StaticRouter action or ContentBasedRouter action you can set the process=”split”, which will enable the Aggregator Action to aggregrate the message further down to processing path.

> Note: that those destinations could be different message transformers to customize the outgoing messages.

Aggregator

The Aggregator action can be used to aggregate message that where split back into 1 message. This message will contain as many message attachments as the messages that where aggregated. The next action in the pipeline should be a transformer action to use the attachments to build a new aggregated message.