JBoss ESB 4.2.1 GA

Content Based Routing

JBESB-CBR-10/31/07

Software Version

JBoss ESB 4.2.1 GA

Contents

Table of Contents

About This Guide

What This Guide Contains

The Content Based Routing contains contain important information on changes to JBoss ESB 4.2.1 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 4.2.1 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 4.2.1 GA documentation set:

1. **JBoss ESB 4.2.1 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.

2. **JBoss ESB 4.2.1 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.

3. **JBoss ESB 4.2.1 GA** *Programmers Guide*: How to use JBossESB.

4. **JBoss ESB 4.2.1 GA** *Release Notes*: Information on the differences between this release and previ ous releases.

5. **JBoss ESB 4.2.1 GA** *Administration Guide*: How to manage the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.2.1 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.

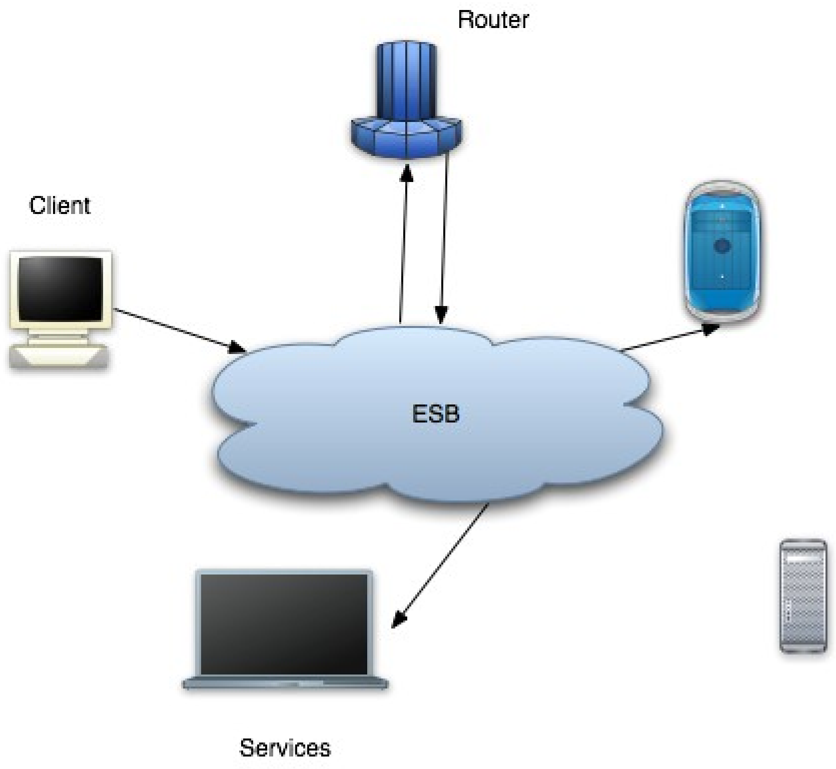
Simple example

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 using Drools

Introduction

The Content Based Router (CBR) in the JBossESB uses JBossRules/Drools as its evaluation engine. JBossESB integrates with Drools through

- three different routing action classes,
- a routing rule set, written in Drools drl (and optionally dsl) language.
- The EsbMessage content, either the serialized XML, or objects in the message, which is the data going into the rules engine.
- destination(s) which is the result coming out of the rules engine.

When a message gets send to the CBR, a certain rule set will evaluate the message content and return a set of Service destinations. We discuss how a target rule set can be targeted, how the message content is evaluated and what is done with the destination results.

Three different routing action classes

JBossESB ships with three slightly different routing action classes. Each of these action classes implements a Enterprise Integration Pattern. For more information of the Enterprise Integration Pattern you can check the JBossESB Wiki. The following actions are supported:

org.jboss.soa.esb.actions.ContentBasedRouter

Implements the Content Based Routing pattern. It routes a message to one or more destination services based on the message content and the rule set it is evaluating it against. The CBR throws an exception when *no* destinations are matched for a given rule set/message combination. This action will terminate any further pipeline processing, so it should the last action of your pipeline.

org.jboss.soa.esb.actions.ContentBasedWireTap

Implements the WireTap pattern. The WireTap is a Enterprise Integration Pattern (EIP) where a copy of the message is send to a control channel. The CBR-WT is identical in functionality to the ContentBasedRouter, however it does *not* terminate the pipeline which makes it suitable to be used as a WireTap.

org.jboss.soa.esb.actions.MessageFilter.

Implements the Message-Filter pattern. The Message Filter pattern represents the case where messages can simply be dropped if certain content requirements are not met. The CBR-MF is identical in functionality to the ContentBasedRouter, but it does *not* throw an exception if the rule set does not match any destinations. In this case the message is simply filter out.

Rule Set Creation

A rule set can be created using the JBossIDE or Red Hat Developer Studio which includes a plug-in for JBossRules. Figure 1 shows a screen shot the plug-in. For a detailed discussion on rule creation and the Drools language itself please see the Drools documention. To turn a regular ruleSet into a Countent Based Routing RuleSet you must be evaluating an EsbMessage and the rule match should result in a creating a List of Strings containing the service destination names. To do this you need to make sure to remember two things:

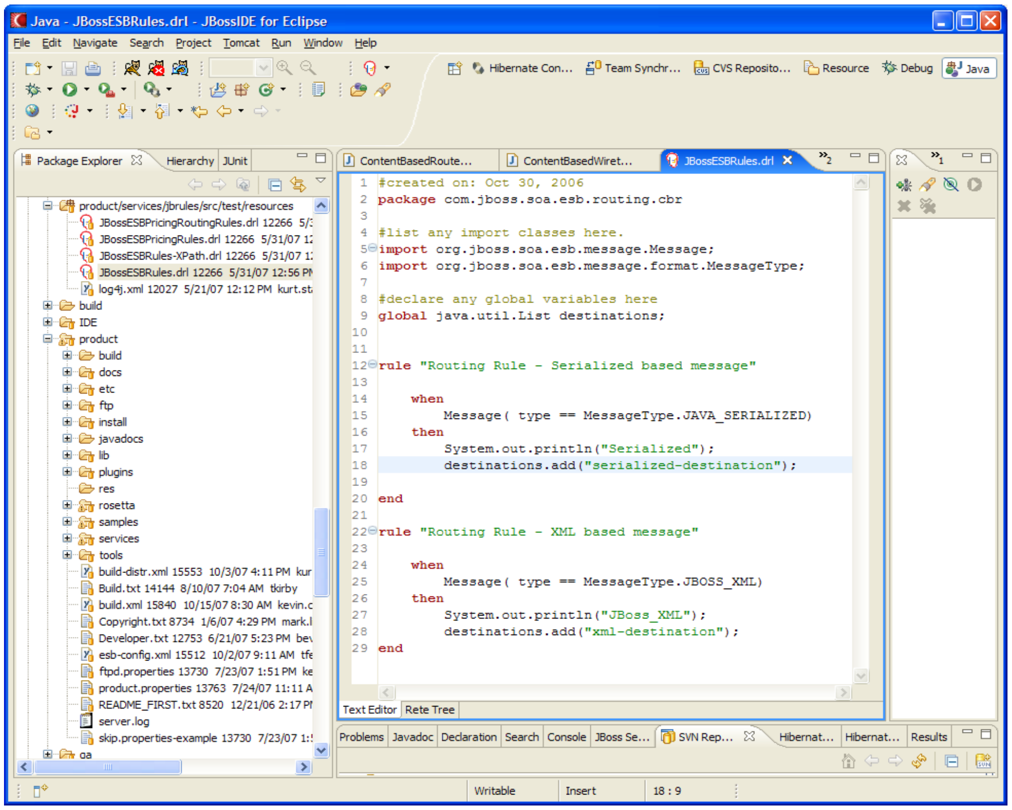
- your rule set imports the EsbMessage

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

- andyour rule set defines

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

  which will make the list of destinations available to the ESB


Java - JBossESBRules.drl - JBossIDE for Eclipse
File Edit Navigate Search Project Tomcat Run Window Help
Hibernate Con... Team Synchr... CVS Reposito... Resource Debug Java
Package Explorer Hierarchy JUnit
ContentBasedRoute... ContentBasedWiret... JBossESBRules.drl
Text Editor Rete Tree
Problems Javadoc Declaration Search Console JBoss Se... SVN Rep... Hibernat... Hibernat... Results
Writable Insert 18 : 9

```
product/services/jbrules/src/test/resources
    JBossESBPricingRoutingRules.drl 12266 5/3
    JBossESBPricingRules.drl 12266 5/31/07 12
    JBossESBRules-XPath.drl 12266 5/31/07 12
    JBossESBRules.drl 12266 5/31/07 12:56 PM
    log4j.xml 12027 5/21/07 12:12 PM kurt.st
build
IDE
product
    build
    docs
    etc
    ftp
    install
    javadocs
    lib
    plugins
    res
    rosetta
    samples
    services
    tools
    build-distr.xml 15553 10/3/07 4:11 PM kur
    Build.txt 14144 8/10/07 7:04 AM tkirby
    build.xml 15840 10/15/07 8:30 AM kevin.c
    Copyright.txt 8734 1/6/07 4:29 PM mark.l
    Developer.txt 12753 6/21/07 5:23 PM bev
    esb-config.xml 15512 10/2/07 9:11 AM tfe
    ftpd.properties 13730 7/23/07 1:51 PM ke
    product.properties 13763 7/24/07 11:11 A
    README_FIRST.txt 8520 12/21/06 2:17 PM
    server.log
    skip.properties-example 13730 7/23/07 1:5
qa
```

```
#created on: Oct 30, 2006
package com.jboss.soa.esb.routing.cbr

#list any import classes here.
import org.jboss.soa.esb.message.Message;
import org.jboss.soa.esb.message.format.MessageType;

#declare any global variables here
global java.util.List destinations;


rule "Routing Rule - Serialized based message"

    when
        Message( type == MessageType.JAVA_SERIALIZED)
    then
        System.out.println("Serialized");
        destinations.add("serialized-destination");

end

rule "Routing Rule - XML based message"

    when
        Message( type == MessageType.JBOSS_XML)
    then
        System.out.println("JBoss_XML");
        destinations.add("xml-destination");
end
```

Figure 1. Create a new ruleSet using JbossIDE or Red Hat Developer Studio

The message will be asserted into the working memory of the rules engine. Figure 2 shows an example where the MessageType is used to determine to which destination the Message is going to be send. This particular ruleSet is shipped in the JBossESBRules.drl file and the rule checks if the type is XML or Serializable.

XPath Domain Specific Language

For XML-based messages it is convenient to do XPath based evaluation. To support this we ship a “Domain Specific Language” implementation which allows us to use XPath expressions in the rule file. defined in the XPathLanguage.dsl. To use it you need to reference it in your ruleSet with:

```
expander XPathLanguage.dsl
```

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.

3. xpathGreaterThan “<element>”, “<value>”): yields true if the element is found and it's value is greater than the val ue.

4. xpathLowerThan “<element>”, “<value>”): yields true if the element is found and it's value is lower then the value.

The XPathLanguage.dsl is defined in a file called XPathLanguage.dsl, and can be customized if needed, or you can define your own DSL altogether. The Quickstart called `fun_cbr` demonstrates this use of XPath.

Configuration

Now that we have seen all the individual pieces how does it all tie together? It basically all comes down to configuration at this point, which is all done in your jboss-esb.xml. Figure 1 shows a service configuration fragment. In this fragment the service is listening on a JMS queue.

Each EsbMessage is passed on to in this case the ContentBasedRouter action class which is loaded with the a certain rule set. It sets the EsbMessage into Working Memory, fires the rules, obtains the list of destinations and routes copies of the EsbMessage to these services. It uses the rule set JbossESBRules.drl, which matches two destinations, name 'xml-destination' and 'serialized-destination'. These names are mapped to real service names in the '`route-to`' section.

```
<service
    category="MessageRouting"
    name="YourServiceName"
    description="CBR Service">
    <listeners>
       <jms-listener name="CBR-Listener"
                  busidref="QueueA" maxThreads="1">
        </jms-listener>
    </listeners>
    <actions>
         <action class="org.jboss.soa.esb.actions.ContentBasedRouter"
                name="YourActionName">
           <property name="ruleSet" value="JBossESBRules.drl"/>
           <property name="ruleReload" value="true"/>
           <property name="destinations">
                <route-to destination-name="xml-destination"
                       service-category="category01"
                       service-name="jbossesbtest1" />
                <route-to destination-name="serialized-destination"
                       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>
```

Figure 2. Example Content Based Routing Service configuration.

The action attributes to the action tag are show in Table 1. The attributes specify which action is to be used and which name this action is to be given.

Attribute	***Description***

Class	Action class, one of :
	`org.jboss.soa.esb.actions.ContentBasedRouter`
	`org.jboss.soa.esb.actions.ContentBasedWireTap`
	`org.jboss.soa.esb.actions.MessageFilter`

Name	Custom action name

Table 1. CBR action configuration attributes.

The action properties are shown in Table 2. The properties specify the set of rules (ruleSet) to be used in this action.

Property	***Description***
ruleSet	Name of the filename containing the Drools ruleSet. The set of rules that is used to evaluate the content. Only 1 ruleSet can be given for each CBR instance.

ruleLanguage	Optional reference to a file containing the definition of a Domain Specific Language to be used for evaluating the rule set.
ruleReload	Optional property which can be to true to enable 'hot' redeployment of rule sets. Note that this feature will cause some overhead on the rules processing. Note that rules will also reload if the .esb archive in which they live is redeployed.
destinations	A set of route-to properties each containing the logical name of the destination along with the Service category and name as referenced in the registry. The logical name is the name which should be used in the rule set.
object-paths	Optional property to pass Message objects into Drools WorkingMemory.

Table 2. CBR action configuration properties.

Object Paths

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? An the optional 'object-paths' property can be used, which results in the extraction of objects from the message, using an “ESB Message Object Path”. MVEL is used to extract the object and the path used should follow the syntax:

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

where,

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, gets getOrder() return object on the attached Object.

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.

It is important to remember that you have to add java import statements on the objects you import into your rule set. Finally, the Object Mapper cannot flatten out entire collections, so if you need to do that you have to a (Smooks-) transformati on on the message first, to unroll the collecti on.

Executing Business Rules

Related to rule execution for routing is the rule execution to simply *modifying* data in the message according to business rules. An example Quickstart called business_rule_service demonstrates this use case. This quickstart uses the action class

```
org.jboss.soa.esb.actions.BusinessRulesProcessor
```

The functionality of the Business Rule Processor (BRP) is identical to the Content Based Router, but it does *not* do any routing, but instead returns the modified EsbMessage for furter action pipeline processing. You may mix business and routing rules in one rule set if you wish to do so, but routing will only occur if you use one of the three routing action classes mentioned earlier.

Deployment and Packaging

It is recommended that you package up your code into units of functionality, using .esb packages. The idea is to package up your routing rules alongside the rule services that use the rule sets. Figure 3 shows a layout of the simple_cbr quickstart to demonstrate a typical package.

```
simple_cbr.esb
|   jbm-queue-service.xml
|   SimpleCBRRules-XPath.drl
|   SimpleCBRRules.drl
|
+---META-INF
|       deployment.xml
|       jboss-esb.xml
|       MANIFEST.MF
|
\---org
    \---jboss
        \---soa
            \---esb
                \---samples
                    \---quickstart
                        \---simplecbr
                            |   MyJMSListenerAction.class
                            |   ReturnJMSMessage.class
                            |   RouteExpressShipping.class
                            |   RouteNormalShipping.class
                            |
                            \---test
                                    ReceiveJMSMessage$1.class
                                    ReceiveJMSMessage.class
                                    SendJMSMessage.class
```

Figure 3. Typical .esb archive which uses Drools.

Finally make sure to deploy and reference the jbrules.esb in your deployment.xml.

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