

ESB Content Based Routing Guide

4.2

JBoss Enterprise SOA Platform



ISBN:

Publication date: February, 2008

The SOA platform edition of the JBoss ESB Content Based Routing Guide

ESB Content Based Routing Guide: JBoss Enterprise SOA Platform

Copyright © 2008 Red Hat, Inc.

Copyright © 2008 Red Hat, Inc.. This material may only be distributed subject to the terms and conditions set forth in the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License (which is presently available at <http://creativecommons.org/licenses/by-nc-sa/3.0/>).

Red Hat and the Red Hat "Shadow Man" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive
Raleigh, NC 27606-2072
USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park, NC 27709
USA

Preface	vii
1. Document Conventions	vii
2. We Need Feedback	viii
1. What is Content Based Routing?	1
1. Introduction	1
1.1. Simple Example	1
2. Content Based Routing using Drools	3
1. Introduction	3
2. Three different routing action classes	3
3. Rule Set Creation	4
4. XPath Domain Specific Language	5
5. Configuration	5
6. Object Paths	7
7. Executing Business Rules	8
8. Deployment and Packaging	8
A. Revision History	11

Preface

1. Document Conventions

Certain words in this manual are represented in different fonts, styles, and weights. This highlighting indicates that the word is part of a specific category. The categories include the following:

Courier font

Courier font represents `commands, file names and paths, and prompts`.

When shown as below, it indicates computer output:

```
Desktop      about.html    logs          paulwesterberg.png
Mail         backupfiles   mail          reports
```

Courier font

Bold Courier font represents text that you are to type, such as: `service jonas start`

If you have to run a command as root, the root prompt (`#`) precedes the command:

```
# gconftool-2
```

italic Courier font

Italic Courier font represents a variable, such as an installation directory:

```
install_dir/bin/
```

font

Bold font represents **application programs** and **text found on a graphical interface**.

When shown like this: **OK**, it indicates a button on a graphical application interface.

Additionally, the manual uses different strategies to draw your attention to pieces of information. In order of how critical the information is to you, these items are marked as follows:



Note

A note is typically information that you need to understand the behavior of the system.



Tip

A tip is typically an alternative way of performing a task.



Important

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.



Caution

A caution indicates an act that would violate your support agreement, such as recompiling the kernel.



Warning

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

2. We Need Feedback

If you find a typographical error in the *ESB Content Based Routing Guide*, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in JIRA: <http://jira.jboss.com/jira/> against the component *FIXME*.

When submitting a bug report, be sure to mention the manual's identifier:

ESB_CBR

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

What is Content Based Routing?

1. Introduction

Information within 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, there are a number of real-world scenarios where this arrangement will fail to deliver the message to the intended recipient. What happens if the specified address is no longer valid - for example, if the service has failed or been removed from service? It is also possible that the service has been changed and 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 now interested in the message's content? What if no destination is specified?

One powerful and flexible solution to these problems is *Content-based Routing*. Content-based routing routes the message based on the content of the message, rather than by an explicitly specified destination. The JBoss ESB Content-based Router opens the message up and applies a set of rules to its content to determine the parties interested in its content. This can free up sending applications from having to know anything about where a message is going to end up, as well as providing a high degree of flexibility and adaptability to change.

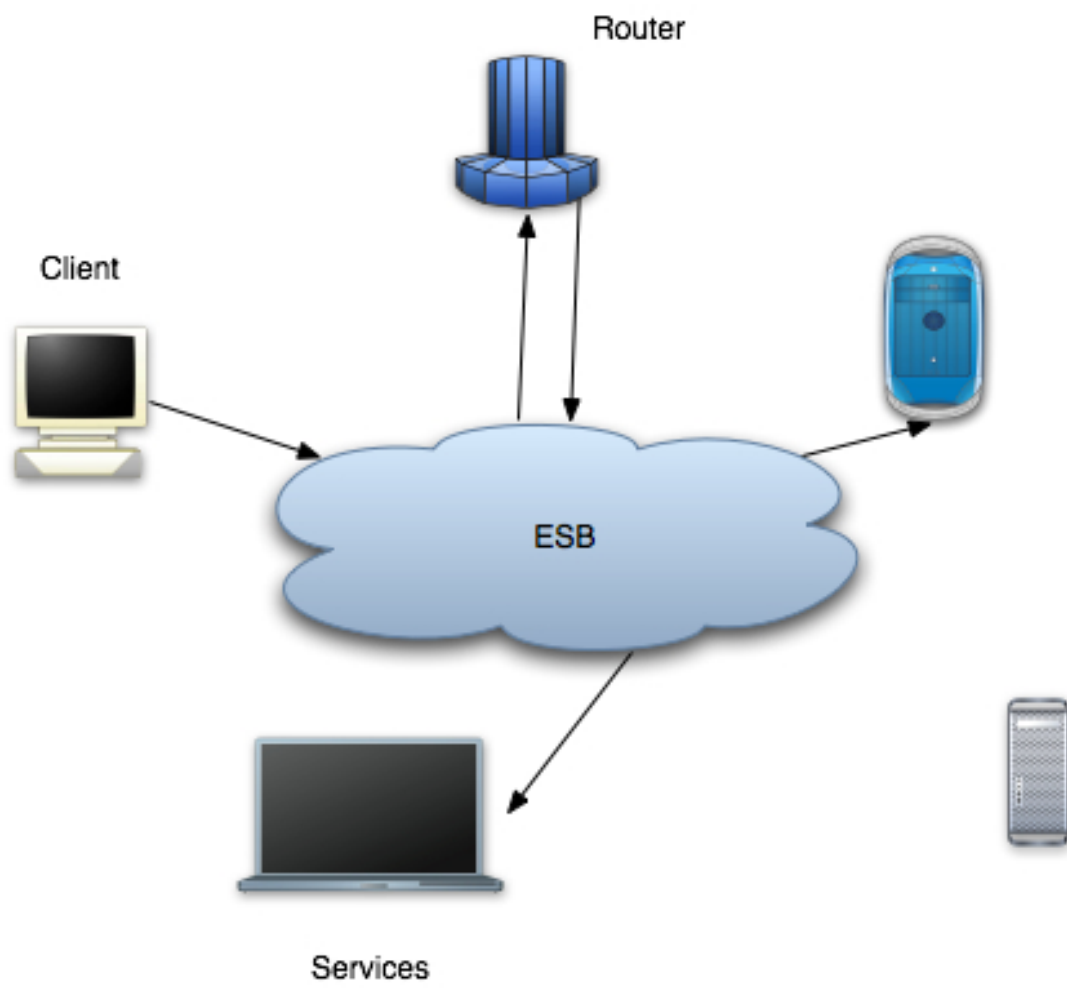
1.1. Simple Example

The Content-based routing system is 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 the Content-based Routing architecture. 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. The Router is then responsible for sending the messages to their ultimate destination (or destinations, as shown in this example).



Content Based Routing using Drools

1. Introduction

The JBossESB Content Based Router (CBR) 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) resulting from rules engine processing.

When a message is sent to the CBR, a particular rule set evaluates the message content and returns a set of Service destinations. We will now examine how a rule set is targeted, how the message content is evaluated, and how the destination results are applied.

2. Three different routing action classes

JBossESB ships with three different routing action classes. Each of these action classes implements an Enterprise Integration Pattern. For more information of the Enterprise Integration Pattern you can check the [JBossESB Wiki](http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBEIP) [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBEIP]. 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 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 be the last action of your pipeline.

`org.jboss.soa.esb.actions.ContentBasedWiretap`

Implements the `WireTap` pattern. The `WireTap` is an Enterprise Integration Pattern (EIP) where a copy of the message is sent 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

filtered out.

3. Rule Set Creation

A rule set can be created using the JBossIDE or JBoss Developer Studio, which include a plug-in for JBossRules. [Figure 2.1, “Create a new ruleSet using JBossIDE or JBoss Developer Studio”](#) shows a screen shot of the plug-in. For a detailed discussion on rule creation and the Drools language itself please see the Drools documentation. To turn a regular RuleSet into a Content Based Routing RuleSet you must be evaluating an ESBMessage and the rule match should result in the creation of 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 class `org.jboss.soa.esb.message.Message`
- and your rule set defines `global java.util.List destinationServices;`, which will make the list of destinations available to the ESB

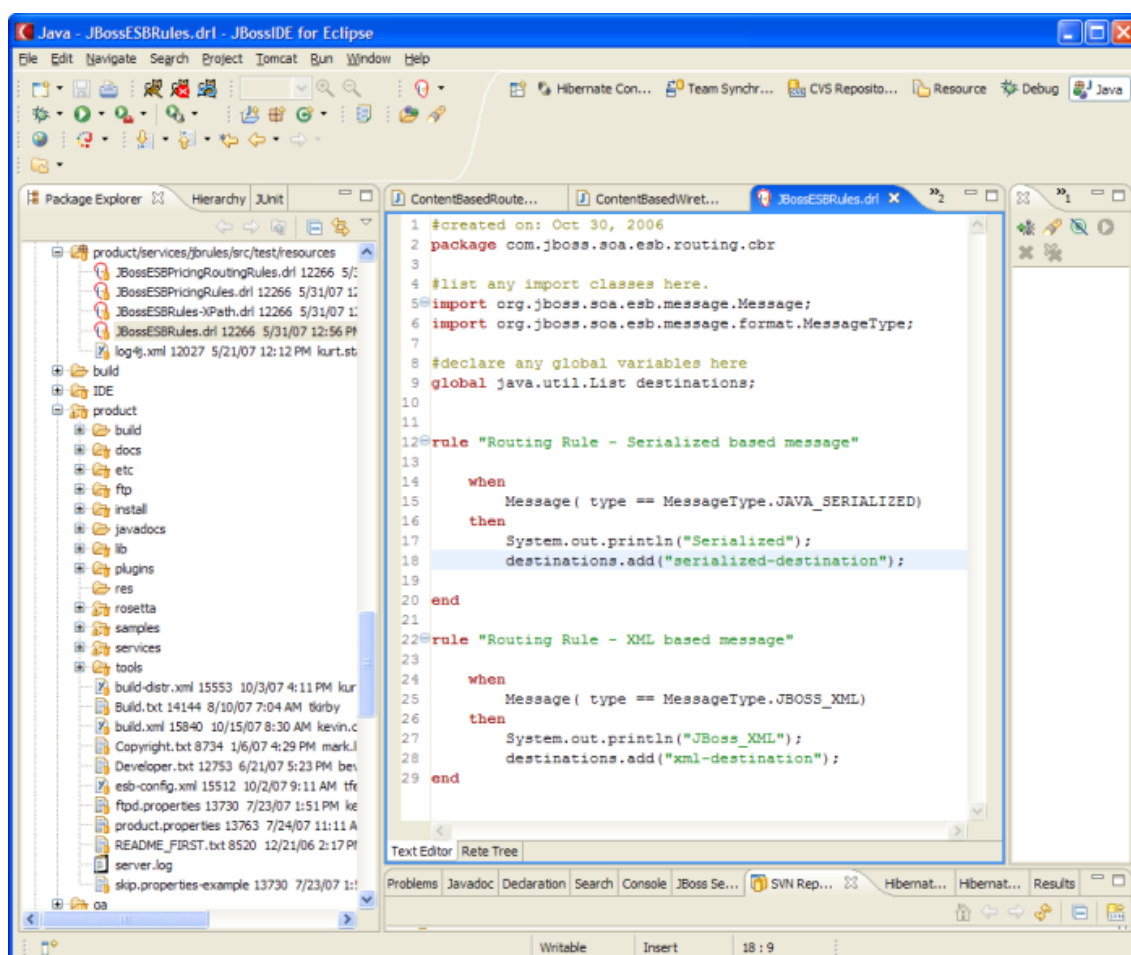


Figure 2.1. Create a new ruleSet using JBossIDE or JBoss Developer Studio

The message will be asserted into the working memory of the rules engine. The xml fragment in [Section 5, “ Configuration ”](#) shows an example where the `MessageType` is used to determine the destination the Message should be sent to. This particular ruleSet is shipped in the `JBossESBRules.drl` file and the rule checks if the type is `XML` or `Serializable`.

4. 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 `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:

`xpathMatch “<element>”`

yields true if an element by this name is matched.

`xpathEquals “<element>”, “<value>”`

yields true if the element is found and its value equals the value.

`xpathGreaterThan “<element>”, “<value>”`

yields true if the element is found and its value is greater than the value.

`xpathLowerThan “<element>”, “<value>”`

yields true if the element is found and its value is lower then the value.

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

5. Configuration

The individual pieces are all tied together through the configuration in `jboss-esb.xml`. The xml fragment below shows a typical service configuration.

In this fragment the service is listening on a JMS queue. Each `ESBMessage` is passed to the `ContentBasedRouter` action class which is loaded with a certain rule set. It loads 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, named `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>

```

The action attributes of the action tag are shown in [Table 2.1, “CBR action configuration attributes”](#). The attributes specify which action to use, and the name to be given to that action.

Attribute	Description
class	<p>Action class. One of:</p> <ul style="list-style-type: none"> org.jboss.soa.esb.actions.ContentBasedRouter org.jboss.soa.esb.actions.ContentBasedWireTap org.jboss.soa.esb.actions.MessageFilter
name	Custom action name

Table 2.1. CBR action configuration attributes

The action properties are shown in [Table 2.2, “CBR action configuration properties”](#). The properties specify the set of rules (ruleSet) to use in this action.

Property	Description
ruleSet	Name of the file containing the Drools ruleSet 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

Property	Description
	definition of a Domain Specific Language to be used for evaluating the rule set.
ruleReload	Optional property which can be set to true to enable 'hot' redeployment of rule sets. Note that this feature will cause some overhead on the rules processing. Note also that rules will automatically reload if the <code>.esb</code> archive in which they live is redeployed.
destinations	A set of <code>route-to</code> 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 Working Memory.

Table 2.2. CBR action configuration properties

6. Object Paths

Note that JBossRules treats objects as shallow objects to achieve highly optimized performance. To evaluate an object deeper than the object tree 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 do a (Smooks-) transformation on the message first, to unroll the collection.

7. Executing Business Rules

Related to rule execution for routing is rule execution that modifies the data in the message according to Business Rules. An example Quickstart called `business_rules_service` demonstrates this. This quickstart uses the action class `org.jboss.soa.esb.actions.BusinessRulesProcessor`.

The Business Rules Processor (BRP) functions much like the Content Based Router, with the important exception that it does not do any routing. Instead it returns the modified `ESBMessage` for further 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.

8. Deployment and Packaging

The recommended practice is to package up code into units of functionality, using `.esb` packages. Routing rules can then be packaged alongside the rule services that use the rule sets. Below is the 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
```

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

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

Appendix A. Revision History

Revision History

Revision 1.0

