JBoss ESB 4.2 Milestone Release 2

Message Action Guide

JBESB-MAG-5/8/07

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 Milestone Release 2

Restricted Rights Legend

Contents

About This Guide

What This Guide Contains

The goal of this document is to:

1. Provide a catalog of all Message Action implementations provided with JBoss ESB (out-of-the-box).

2. Provide a guide for developing custom Action implementations.

Audience

This guide is targeted at developers.

Prerequisites

None.

Organization

See document index.

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 Milestone Release 2 documentation set:

1. **JBoss ESB 4.2 Milestone Release 2** *Getting Started Guide*: Quick guide to getting started with JBoss ESB..
2. **JBoss ESB 4.2 Milestone Release 2** *Programmers Guide*: How to use JBossESB.
3. **JBoss ESB 4.2 Milestone Release 2** *Administration Guide*: How to manage the ESB.
4. **JBoss ESB 4.2 Milestone Release 2** *Services Guides*: Various documents related to the services available with the ESB.
5. **JBoss ESB 4.2 Milestone Release 2** *Trailblazer Guide*: Provides guidance for using the trailblazer example.
6. **JBoss ESB 4.2 Milestone Release 2** *Release Notes*: Information on the differences between this release and previous releases.

Contacting Us

Questions or comments about JBoss ESB 4.2 Milestone Release 2 should be directed to our support team.

Chapter 1

Pre-Packed Actions

This section provides a catalog of all Actions that are supplied out-of-the-box with JBoss ESB ("pre-packed").

Transformers & Converters

Converters/Transformers are a classification of Action Processor responsible for transforming a message (payload, headers, attachments etc) from a format produced by one message exchange participant, into a format that is consumable by anopther message exchange participant.

ByteArrayToString

Takes a ***byte[]*** based message payload and converts it into a *java.lang.String* object instance, bound to the message under the name "*org.jboss.soa.esb.actions.current.after*".

Input Type	byte[]
Input Location	• Body.contents • Body."*org.jboss.soa.esb.actions.current.after*"
Output Type	java.lang.String
Output Location	• Body."*org.jboss.soa.esb.actions.current.after*"
Class	org.jboss.soa.esb.actions.converters.ByteArrayToString
Properties	• "***encoding***": The binary data encoding on the message byte array. Defaults to "UTF-8" when not specified .

Sample Config	`<action name="transform"`
	`class="org.jboss.soa.esb.actions.converters.ByteArrayToString">`
	
	`<property name="encoding" value="UTF-8" />`
	`</action>`

LongToDateConverter

Takes a ***long*** based message payload and converts it into a *java.util.Date* object instance, bound to the message under the name "*org.jboss.soa.esb.actions.current.after*".

Input Type	java.lang.Long/long
Input Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Output Type	java.util.Date
Output Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Class	org.jboss.soa.esb.actions.converters.LongToDateConverter
Properties	None
Sample Config	`<action name="transform"` `  class="org.jboss.soa.esb.actions.converters.LongToDateConverter" />`

ObjectInvoke

Takes the Object bound to a message under the name "*org.jboss.soa.esb.actions.current.after*" and supplies it to a configured “processor” for processing. The processing result is bound to the message under the name "*org.jboss.soa.esb.actions.current.after*" (overwriting the input parameter).

Input Type	User Object
Input Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Output Type	User Object
Output Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Class	org.jboss.soa.esb.actions.converters.ObjectInvoke
Properties	• "***class-processor***": The runtime class name of the processor class used to process the message payload. • "***class-method***": The name of the method on the processor class used to process the method.

Sample Config	`<action name="invoke"`
	`  class="org.jboss.soa.esb.actions.converters.ObjectInvoke">`
	
	`  <property name="class-processor" value="org.jboss.MyXXXProcessor" />`
	`  <property name="class-method" value="processXXX" />`
	`</action>`

ObjectToCSVString

Takes the Object bound to a message under the name "*org.jboss.soa.esb.actions.current.after*" and converts it into a Comma Separated Value (CSV) String based on the supplied message object and a comma-separated "bean-properties" list property.

Input Type	User Object
Input Location	• Body."*org.jboss.soa.esb.actions.current.after*"
Output Type	java.lang.String
Output Location	• Body."*org.jboss.soa.esb.actions.current.after*"
Class	org.jboss.soa.esb.actions.converters.ObjectToCSVString
Properties	• "***bean-properties***": List of Object bean property names used to get CSV values for the output CSV String. The Object should support a getter method for each of listed properties. • "***fail-on-missing-property***": Flag indicating whether or not the action should fail if a property is missing from the Object i.e. If the Object doesn't support a getter method for the property. Default value is "false".

Sample Config	`<action name="transform"`
	`class="org.jboss.soa.esb.actions.converters.ObjectToCSVString">`
	
	`<property name="bean-properties" value="name,address,phoneNumber" />`
	`<property name="fail-on-missing-property" value="true" />`
	`</action>`

ObjectToXStream

Takes the Object bound to a message under the name "*org.jboss.soa.esb.actions.current.after*" and converts it into XML using the XStream processor.

Input Type	User Object
Input Location	• Body."*org.jboss.soa.esb.actions.current.after*"
Output Type	java.lang.String
Output Location	• Body."*org.jboss.soa.esb.actions.current.after*"
Class	org.jboss.soa.esb.actions.converters.ObjectToXStream
Properties	• "***class-alias***": Class alias used in call to XStream.alias(String, Class) prior to serialisation. Defaults to the input Object's class name. • "***exclude-package***": Exclude the package name from the generated XML. Default is "true". Not applicable if a "class-alias" is specified.

Sample Config	<action name="transform" class="org.jboss.soa.esb.actions.converters.ObjectToXStream"> <property name="class-alias" value="MyAlias" /> <property name="exclude-package" value="true" /> </action>

SmooksTransformer

Performs a message transformation based on the specified message exchange.

Input Type	java.lang.String
Input Location	• Body.contents • Body.”*org.jboss.soa.esb.actions.current.after*”
Output Type	java.lang.String (User Objects, where beans populated by Smooks Javabean based transforms)
Output Location	• Body.”*org.jboss.soa.esb.actions.current.after*” • Body.”EXTRACTED_BEANS_HASH” (for beans populated by Smooks Javabean based transforms).
Class	org.jboss.soa.esb.actions.converters.SmooksTransformer
Properties	• "***from***": Message Exchange Participant name. Message Producer. • "***from-type***": Message type/format produced by the “from” message exchange participant. • "***to***": Message Exchange Participant name. Message Consumer. • "***to-type***": Message type/format consumed by the “to” message exchange participant. Note: All the above properties can be overridden by supplying them as properties to the message (Message.Properties).
Sample Config	<action name="transform" class="org.jboss.soa.esb.actions.converters.SmooksTransformer"> <property name="from" value="DVDStore:OrderDispatchService" /> <property name="from-type" value="text/xml:fullFillOrder" /> <property name="to" value="DVDWarehouse_1:OrderHandlingService" /> <property name="to-type" value="text/xml:shipOrder" /> </action>

See the MessageTransformation.pdf for more details on the SmooksTransformer.

XStreamToObject

Takes the XML bound to a message under the name "*org.jboss.soa.esb.actions.current.after*" and converts it into an Object using the XStream processor.

Input Type	java.lang.String
Input Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Output Type	User Object (specified by “incoming-type” property)
Output Location	• Body.”*org.jboss.soa.esb.actions.current.after*”
Class	org.jboss.soa.esb.actions.converters.XStreamToObject
Properties	• "***class-alias***": Class alias used during serialisation. Defaults to the input Object's class name. • "***exclude-package***": Flag indicating whether or not the XML includes a package name. • "***incoming-type***": Class type.

Sample Config	`<action name="transform"`
	`  class="org.jboss.soa.esb.actions.converters.XStreamToObject">`
	
	`  <property name="class-alias" value="MyAlias" />`
	`  <property name="exclude-package" value="true" />`
	`  <property name="incoming-type" value="com.acme.MyXXXClass" />`
	`</action>`

jBPM - CommandInterpreter

Expects the argument to be a command message and tries to execute the corresponding jBPM api invocation. If Call in message header contains a replyToEpr, will send response to it.

jBPM configuration files (jbpm.cfg.xml and hibernate.cfg.xml) must be present where the Jbpm.Configuration.getInstance() expects them to be found.

At present time, the following operations are implemented :

deployProcessDefinition
,newProcessInstance
,signalProcess
,signalToken
,getProcessInstanceVariables
,setProcessInstanceVariables
,getTokenVariables
,setTokenVariables
,hasInstanceEnded

Input Type	org.jboss.soa.esb.message.Message generated by AbstractCommandVehicle.toCommandMessage()
Input Location	• full message
Output Type	Message – output of util.jbpm.CommandVehicle.toCommandMessage() containing result of jBPM api call
Output Location	• full message
Class	org.jboss.soa.esb.actions.jbpm.CommandInterpreter
Properties	•

Sample Config	`<action name="process"`
	`class="org.jboss.soa.esb.actions.jbpm.CommandInterpreter">`
	`</action>`

Scripting

Scripting Action Processors support definition of action processing logic via Scripting languages.

GroovyActionProcessor

Executes a Groovy action processing script, receiving the message and action configuration as input.

Script Bindings	• "***message***": The message. • "***config***": The action configuration (ConfigTree).
Class	org.jboss.soa.esb.actions.scripting.GroovyActionProcessor
Properties	• "***script***": Path (classpath) to Groovy script.

Sample Config	`<action name="process"`
	`  class="org.jboss.soa.esb.scripting.GroovyActionProcessor">`
	
	`  <property name="script" value="/scripts/ActionXProcessor.groovy" />`
	`</action>`

Routing

Routing Actions support conditional routing of messages between two or more message exchange participants.

Aggregator

Message aggregation action. An implementation of the Aggregator Enterprise Integration Pattern.

Class	org.jboss.soa.esb.actions.Aggregator
Properties	• "***timeoutInMillies***": Timeout time in milliseconds before the aggregation process times out.

Sample Config	`<action class="org.jboss.soa.esb.actions.Aggregator" name="Aggregator">`
	`  <property name="timeoutInMillies" value="60000"/>`
	`</action>`

This action relies on all messages having the correct correlation data. This data is set on the message as a property called "aggregatorTag" (Message.Properties). See the ContentBasedRouter and StaticRouter actions.

The data has the following format:

```
[UUID] ":" [message-number] ":" [message-count]
```

If all the messages have been received by the aggregator, it returns a new Message containing all the messages as part of the Message.Attachment list (unnamed), otherwise the action returns null.

ContentBasedRouter

Content (plus rules) based message routing action.

Class	org.jboss.soa.esb.actions.ContentBasedRouter
Properties	• "***ruleSet***": JBoss Rules ruleset. • "***ruleLanguage***": CBR evaluation Domain Specific Language (DSL) file. • "***ruleReload***": Flag indicating whether or not the rules file should be reloaded each time. Default is "false". • "***destinations***": Container property for the <route-to> configurations. ➢ `<route-to destination-name="express" service-category="ExpressShipping" service-name="ExpressShippingService"/>`
"process" methods	• "***process***": Don't append aggregation data to message. • "***split***": Append aggregation data to message. See the Aggregator action.

Sample Config	`<action process="split" name="ContentBasedRouter" class="org.jboss.soa.esb.actions.ContentBasedRouter">`
	`<property name="ruleSet" value="MyESBRules-XPath.drl"/>`
	`<property name="ruleLanguage" value="XPathLanguage.dsl"/>`
	`<property name="ruleReload" value="true"/>`
	`<property name="destinations">`
	`<route-to destination-name="express" service-category="ExpressShipping" service-name="ExpressShippingService"/>`
	`<route-to destination-name="normal" service-category="NormalShipping" service-name="NormalShippingService"/>`
	`</property>`
	`</action>`

See ContentBasedRouting.pdf for more details on the Content Based Routing.

StaticRouter

Static message routing action. This is basically a simplified version of the Content Based Router, accept it doesn't support content based routing rules.

Class	org.jboss.soa.esb.actions.ContentBasedRouter
Properties	• "***destinations***": Container property for the <route-to> configurations. ➢ `<route-to destination-name="express" service-category="ExpressShipping" service-name="ExpressShippingService"/>`
"process" methods	• "***process***": Don't append aggregation data to message. • "***split***": Append aggregation data to message. See the Aggregator action.

Sample Config	`<action name="routeAction"`
	`class="org.jboss.soa.esb.actions.StaticRouter">`
	`<property name="destinations">`
	`<route-to service-category="ExpressShipping"`
	`service-name="ExpressShippingService"/>`
	`<route-to service-category="NormalShipping"`
	`service-name="NormalShippingService"/>`
	`</property>`
	`</action>`

Notifier

Send notifications to list specified in configuration. This class has a dummy process(Message) method that simply returns the argument.

Intended as example of what's needed to have your own notifier. You would typically extend this class and override notifyOk() and notifyError() methods to produce the desired output.

If you wish the ability to notify of success or failure at each step of the action processing pipeline, use the "okMethod" and "exceptionMethod" attributes in each <action> element instead of having an <action> that uses the Notifier class.

Class	org.jboss.soa.esb.actions.Notifier
Properties	NotificationList subtree indicating targets

Sample Config	`<action class="org.jboss.soa.esb.actions.Notifier" okMethod="notifyOK">`
	`<property name="destinations">`
	`<NotificationList type="OK">`
	`<target class="NotifyConsole" />`
	`</NotificationList>`
	`</property>`
	`</action>`

Miscellaneous

Miscellaneous Action Processors.

SystemPrintln

Simple action for printing out the contents of a message (ala System.out.println).

Will attempt to format the message contents as XML.

Input Type	java.lang.String
Input Location	• Body.contents • Body."*org.jboss.soa.esb.actions.current.after*"
Class	org.jboss.soa.esb.actions.SystemPrintln
Properties	• "***message***": A message prefix.

Sample Config	`<action name="print-before" class="org.jboss.soa.esb.actions.SystemPrintln">`
	`  <property name="message" value="Message before action XXX" />`
	`</action>`

Chapter 2

Developing Custom Actions

To implement a custom Action Processor, simply implement the ***org.jboss.soa.esb.actions.ActionPipelineProcessor*** interface.

This interface supports implementation of stateless actions that have a managed lifecycle. A single instance of a class implementing this interface is instantiated on a per pipeline basis (i.e. per action configuration). This means you can cache resources needed by the action in the ***initialise*** method, and clean them up in the ***destroy*** method.

The implementing class should process the message from within the ***process*** method implementation.

As a convenience, you should simple extend the ***org.jboss.soa.esb.actions.AbstractActionPipelineProcessor***.

Example:

```
public class ActionXXXProcessor extends AbstractActionPipelineProcessor {

    public void initialise() throws ActionLifecycleException {
        // Initialise resources...
    }

    public Message process(final Message message) throws ActionProcessingException {
        // Process messages in a stateless fashion...
    }

    public void destroy() throws ActionLifecycleException {
        // Cleanup resources...
    }
}
```

Configuring Actions Using Properties

Actions generally act as templates that require external configuration to perform their tasks. For example, a PrintMessage action might take a property named 'message' to indicate what to print and a property 'repeatCount' to indicate the number of times to print it. The action configuration in the jboss-esb.xml file might look like this:

```
<action name="PrintAMessage" class="test.PrintMessage">
    <property name="message" value="Hello World!" />
    <property name="repeatCount" value="5" />
</action>
```

The default method for loading property values in an action implementation is the use of a ConfigTree instance. The ConfigTree provides a DOM-like view of the action XML. By default, actions are expected to have a public constructor that takes a ConfigTree as a parameter. For example:

```
public class PrintMessage extends AbstractActionPipelineProcessor {

        private String message;

        private Integer repeatCount;

        public PrintMessage(ConfigTree config) {
            message = config.getAttribute("message");
            repeatCount = new Integer(config.getAttribute("repeatCount"));
        }

        public Message process(Message message) throws
             for (int i=0; i < repeatCount; i++) {
                 System.out.println(message);
             }
        }
}
```

Another approach to setting action properties is to add setters on the action that correspond to the property names and allow the framework to populate them automatically. In order to have the action bean auto-populated, the action class must implement the ***org.jboss.soa.esb.actions.BeanConfiguredAction*** marker interface. For example, the following class has the same behavior as the one above.

```
public class PrintMessage extends AbstractActionPipelineProcessor
    implements BeanConfiguredAction {

        private String message;

        private Integer repeatCount;

        public setMessage(String message) {
            this.message = message;
        }

        public setRepeatCount(Integer repeatCount) {
            this.repeatCount = repeatCount;
        }

        public Message process(Message message) throws
             for (int i=0; i < repeatCount; i++) {
                 System.out.println(message);
             }
        }
}
```

Note that the Integer parameter in setRepeatCount() is automatically converted from the String representation specified in the XML.

The BeanConfiguredAction method of loading properties is a good choice for actions that take simple arguments, while the ConfigTree method is better when you need to deal with the XML representation directly.