JBossESB 4.2.1 GA

Service Orchestration Guide

JBESB-SOG-10/31/07

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.

Software Version

JBossESB 4.2.1 GA

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions as set forth in contract subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause 52.227-FAR14.

Contents

Table of Contents

About This Guide

What This Guide Contains

The Service Orchestration Guide document contains descriptions on the principles behind Service Oriented Architecture and Enterprise Service Bus, as well as how they relate to JBossESB.

Audience

This guide is most relevant to engineers who are responsible for using JBossESB 4.2.1 GA installations and want to know how it relates to SOA and ESB principles.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, Orchestration:** A discussion of how jBPM and WS-BPEL can be used to orchestrate services within 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 JBossESB 4.2.1 GA documentation set:

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

2. **JBossESB 4.2.1 GA** *Programmer's Guide*: Provides guidance for developing applications using JBossESB.

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

4. **JBossESB 4.2.1 GA** *Administration Guide*: How to manage JBossESB.

5. **JBossESB 4.2.1 GA** *Release Notes*: Information on the differences between this release and previous releases.

6. **JBossESB 4.2.1 GA** *Services Guides*: Various documents related to the services available with the ESB.

Contacting Us

Questions or comments about JBossESB 4.2.1 GA should be directed to our support team.

Chapter 1

Orchestrating Services

Introduction

There are several ways in which you can coordinate the interactions between your services and the tasks that individual services may execute in order to perform their work. In this chapter we shall give an overview of them. In the case where 3rd party applications and tools are used, you should consult associated documentation as well.

jBPM

Interoperation with jBPM is now possible using the CommandInterpreter and BaseActionHandler classes in the org.jboss.soa.esb.actions.jbpm package. Both use the org.jboss.soa.esb.util.jbpm.CommandVehicle class as a standard to talk each other. CommandVehicle has a constructor that takes a Message as argument, and inherits the toCommandMessage() method (that serializes the object into a Message) from it's parent class.

jBPM api calls that are available can be found in CommandVehicle.java. CommandInterpreter has now basic functionality, and is intended to grow to include more and more of the jBPM api power. Whenever a new call needs to be implemented we should a) add the operation in the 'Operation' enumeration in CommandVehicle and b) modify the process(Message) method in the CommandInterpreter class to do what's necessary to invoke the jBPM method, and place return values in the reply Message. Sometimes new getters/setters might also be needed in the CommandVehicle class.

The jbpm_simple1 quickstart illustrates how to use the jBPM interface classes to deply a process definition, create a process instance, signal a token (and/or process) through it's states, and at any point check if the process has completed (i.e. if it's in an end state).

The BaseActionHandler class extends jBPM ActionHandler, and can be used to send an ESB Message to a registered service from jBPM. It implements an outgoing only message; future versions will include quasi synchronous two way communication. It assumes that two jBPM context variables are set ('esbCategoryName' and 'esbServiceName'), and will include another context variable in the Message payload. The variable name to be included is rendered by this class' getContentVariableName() method and has a default value of "esbUserObjectVariable". Should users wish to include a different context variable in the message, simply extend this class, override the getContentVariableName() method, and use your class as the jBPM ActionHandler.

Using jBPM from within ESB allows (among several other features) persisting process state and handling timers and wait states; powerful features that are needed in (and essential part of) many business processes and don't seem to fall in the realm of ESB itself.

WS-BPEL

JBossESB provides WS-BPEL support via its Web Service components. For details on these components and how to configure and use them, see the Message Action Guide.

JBoss and JBossESB also have a special support agreement with ActiveEndpoints for their award wining ActiveBPEL WS-BPEL Engine. In support of this, JBossESB ships with a Quickstart dedicated to demonstrating how JBossESB and ActiveBPEL can collaborate effectively to provide a WS-BPEL based orchestration layer on top of a set of Services that don't expose Webservice Interfaces (the "webservice_bpel" Quickstart). JBossESB provides the Webservice Integration and ActiveBPEL provides the Process Orchestration. A number of flash based walk-thrus of this Quickstart are also available online.

> Note: ActiveEndpoints WS-BPEL engine does not run on versions of JBossAS since 4.0.5. However, it can be deployed and run successfully on Tomcat as our examples illustrate.

Index