JBossESB 4.3 GA

Connectors and Adapters Guide

JBESB-CAG-5/20/08

JBESB-CAG-5/20/08

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

JBoss, Home of Professional Open Source Copyright 2006, JBoss Inc., and individual contributors as indicated by the @authors tag. All rights reserved.

See the copyright.txt in the distribution for a full listing of individual contributors. This copyrighted material is made available to anyone wishing to use, modify, copy, or redistribute it subject to the terms and conditions of the GNU General Public License, v. 2.0. This program is distributed in the hope that it will be useful, but WITHOUT A WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details. You should have received a copy of the GNU General Public License, v. 2.0 along with this distribution; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Software Version

JBossESB 4.3 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.

© Copyright 2008 JBoss Inc.

Contents

Table of Contents

About This Guide

What This Guide Contains

The Connectors and Adapters Guide contains descriptions on the principles behind Service Oriented Architecture and Enterprise Service Bus, as well as how they relate to JBossESB. This guide also contains information on how to use JBossESB 4.3 GA.

Audience

This guide is most relevant to engineers who are responsible for using JBossESB 4.3 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,** Gateways: A description of how to interface non-ESB aware services and consumers with JBossESB.
- **Chapter 2, JCA:** How to tie JCA in to 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

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBossESB 4.3 GA documentation set:

1. **JBossESB 4.3 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.
2. **JBossESB 4.3 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.
3. **JBossESB 4.3 GA** *Administration Guide*: How to manage JBossESB.
4. **JBossESB 4.3 GA** *Release Notes*: Information on the differences between this release and previous releases.
5. **JBossESB 4.3 GA** *Services Guides*: Various documents related to the services available with the ESB.

Contacting Us

Questions or comments about JBossESB 4.3 GA should be directed to our support team.

Chapter 1

Connectors and Adapters

Introduction

Not all clients and services of JBossESB will be able to understand the protocols and Message formats it uses natively. As such there is a need to be able to bridge between ESB-aware endpoints (those that understand JBossESB) and ESB-unaware endpoints (those that do not understand JBossESB). Such bridging technologies have existed for many years in a variety of distributed systems and are often referred to as Connectors, Gateways or Adapters.

One of the aims of JBossESB is to allow a wide variety of clients and services to interact. JBossESB does not require that all such clients and services be written using JBossESB or any ESB for that matter. There is an abstract notion of an *Interoperability Bus* within JBossESB, such that endpoints that may not be JBossESB-aware can still be "plugged in to" the bus.

> Note: in what follows, the terms "within the ESB" or "inside the ESB" refer to ESB-aware endpoints.

All JBossESB-aware clients and services communicate with one another using `Messages`, to be described later. A `Message` is simply a standardized format for information exchange, containing a header, body (payload), attachments and other data. Furthemore, all JBossESB-aware services are identified using *Endpoint References* (EPRs), to be described later.

It is important for legacy interoperability scenarios that a SOA infrastructure such as JBossESB allow ESB-unaware clients to use ESB-aware services, or ESB-aware clients to use ESB-unaware services. The concept that JBossESB uses to facilitate this interoperability is through *Gateways*. A gateway is a service that can bridge between the ESB-aware and ESB-unaware worlds and translate to/from `Message` formats and to/from EPRs.

JBossESB currently supports Gateways and Connectors. In the following sections we shall examine both concepts and illustrate how they can be used.

The Gateway

Not all users of JBossESB will be ESB-aware. In order to facilitate those users interacting with services provided by the ESB, JBossESB has the concept of a Gateway: specialised servers that can accept messages from non-ESB clients and services and route them to the required destination.

A Gateway is a specialised listener process, that behaves very similarly to an ESB aware listener. There are some important differences however:

- Gateway classes can pick up arbitrary objects contained in files, JMS messages, SQL tables etc (each 'gateway class' is specialized for a specific transport), whereas JBossESB listeners can only process JBossESB normalized `Messages` as described in "The Message" section of this document. However, those `Messages` can contain arbitrary data.

- Only one action class is invoked to perform the 'message composing' action. ESB listeners are able to execute an action processing pipeline.

- Objects that are 'picked up' will be used to invoke a single 'composer class' (the action) that will return an ESB `Message` object, which will be delivered to a target service that must be an ESB aware service. The target service defined at configuration time, will be translated at runtime into an EPR (or a list of EPRs) by the Registry. The underlying concept is that the EPR returned by the Registry is analogous to the 'toEPR' contained in the header of ESB `Messages`, but because incoming objects are 'ESB unaware' and there is thus no dynamic way to determine the toEPR, this value is provided to the gateway at configuration time and included in all outgoing messages.

There are a few off the shelf composer classes: the default 'file' composer class will just package the file contents into the `Message` body; same idea for JMS messages. Default message composing class for a SQL table row is to package contents of all columns specified in configuration, into a `java.util.Map`.

Although these default composer classes will be enough for most use cases, it is relatively straightforward for users to provide their own message composing classes. The only requirements are a) they must have a constructor that takes a single `ConfigTree` argument, and b) they must provide a '`Message` composing' method (default name is 'process' but this can be configured differently in the 'process' attribute of the <action> element within the `ConfigTree` provided at constructor time. The processing method must take a single argument of type `Object`, and return a `Message` value.

Gateway Data Mappings

When a non-JBossESB message is received by a Gateway it must be converted to a `Message`. How this is done and where in the `Message` the received data resides, depends upon the type of Gateway. How this conversion occurs depends upon the type of Gateway; the default conversion approach is described below:

- JMS Gateway: if the input message is a JMS `TextMessage`, then the associated `String` will be placed in the default named `Body` location; if it is an `ObjectMessage` or a `BytesMessage` then the contents are placed within the `BytesBody.BYTES_LOCATION` named `Body` location.

- Local File Gateway: the contents are placed within the `BytesBody.BYTES_LOCATION` named `Body` location.
- Hibernate Gateway: the contents are placed within the `ListenerTagNames.HIBERNATE_OBJECT_DATA_TAG` named `Body` location.
- Remote File Gateway: the contents are placed within the `BytesBody.BYTES_LOCATION` named `Body` location.

> Note: With the introduction of the InVM transport, it is now possible to deploy services within the same address space (VM) as a gateway, improving the efficiency of gateway-to-listener interactions.

How to change the Gateway Data Mappings

If you want to change how this mapping occurs then it will depend upon the type of Gateway:

- File Gateways: instances of the `org.jboss.soa.esb.listeners.message.MessageComposer` interface are responsible for performing the conversion. To change the default behavior, provide an appropriate implementation that defines your own `compose` and `decompose` methods. The new `MessageComposer` implementation should be provided in the configuration file using the `composer-class` attribute name.
- JMS and Hibernate Gateways: these implementations use a reflective approach for defining composition classes. Provide your own `Message` composer class and use the `composer-class` attribute name in the configuration file to inform the Gateway which instance to use. You can use the composer-process attribute to inform the Gateway which operation of the class to call when it needs a `Message`; this method must take an `Object` and return a `Message`. If not specified, a default name of `process` is assumed.

> Note: Whichever of the methods you use to redefine the `Message` composition, it is worth noting that you have complete control over what is in the `Message` and not just the `Body`. For example, if you want to define `ReplyTo` or `FaultTo` EPRs for the newly created `Message`, based on the original content, sender etc., then you should consider modifying the header too.

Connecting via JCA

You can use JCA Message Inflow as an ESB Gateway. This integration does not use MDBs, but rather ESB's lightweight inflow integration. To enable a gateway for a service, you must first implement an endpoint class. This class is a Java class that must implement the org.jboss.soa.esb.listeners.jca.InflowGateway class:

```
public interface InflowGateway
{
  public void setServiceInvoker(ServiceInvoker invoker);
}
```

The endpoint class must either have a default constructor, or a constructor that takes a ConfigTree parameter. This Java class must also implement the messaging type of the JCA adapter you are binding to. Here's a simple endpoint class example that hooks up to a JMS adapter:

```
public class JmsEndpoint implements InflowGateway, MessageListener
{
   private ServiceInvoker service;
   private PackageJmsMessageContents transformer = new
PackageJmsMessageContents();

   public void setServiceInvoker(ServiceInvoker invoker)
   {
      this.service = invoker;
   }

   public void onMessage(Message message)
   {
      try
      {
         org.jboss.soa.esb.message.Message esbMessage =
transformer.process(message);

         service.postMessage(esbMessage);
      }
      catch (Exception e)
      {
         throw new RuntimeException(e);
      }
   }
}
```

One instance of the JmsEndpoint class will be created per gateway defined for this class. This is not like an MDB that is pooled. Only one instance of the class will service each and every incoming message, so you must write threadsafe code.

At configuration time, the ESB creates a ServiceInvoker and invokes the setServiceInvoker method on the endpoint class. The ESB then activates the JCA endpoint and the endpoint class instance is ready to receive messages. In the JmsEndpoint example, the instance receives a JMS message and converts it to an ESB message type. Then it uses the ServiceInvoker instance to invoke on the target service.

> Note: The JMS Endpoint class is provided for you with the ESB distribution under org.jboss.soa.esb.listeners.jca.JmsEndpoint It is quite possible that this class would be used over and over again with any JMS JCA inflow adapters.

Configuration

A JCA inflow gateway is configured in a jboss-esb.xml file. Here's an example:

```
...
      <service category="HelloWorld_ActionESB"
               name="SimpleListener"
               description="Hello World">
         <listeners>
            <jca-gateway name="JMS-JCA-Gateway"
                         adapter="jms-ra.rar"
                         endpointClass="org.jboss.soa.esb.listeners.
jca.JmsEndpoint">
                  <activation-config>
                     <property name="destinationType"
value="javax.jms.Queue"/>
                     <property name="destination"
value="queue/esb_gateway_channel"/>
                  </activation-config>
            </jca-gateway>
...
      </service>
```

JCA gateways are defined in <jca-gateway> elements. These are the configurable attributes of this XML element.

Attribute	Required	Description
name	yes	The name of the gateway
adapter	yes	The name of the adapter you are using. In JBoss it is the filename of the RAR you deployed, e.g., jms-ra.rar
endpointClass	yes	The name of your endpoint class
messagingType	no	The message interface for the adapter. If you do not specify one, ESB will guess based on the endpoint class.
transacted	no	Default to true. Whether or not you want to invoke the message within a JTA transaction.

You must define an <activation-config> element within <jca-gateway>. This element takes one or more <property> elements which have the same syntax as action properties. The properties under <activation-config> are used to create an activation for the JCA adapter that will be used to send messages to your endpoint class. This is really no different than using JCA with MDBs.

You may also have as many <property> elements as you want within <jca-gateway>. This option is provided so that you can pass additional configuration to your

endpoint class. You can read these through the ConfigTree passed to your constructor.
