

# **JBoss ESB 4.2 GA**

---

## Trailblazer Guide

JBESB-TB-8/29/07



## **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**

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**

#### **JBoss ESB 4.2 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 2007 JBoss Inc.

# Contents

## Table of Contents

<b>Contents.....</b>	<b>iii</b>	Contacting Us.....	v
.....	iii	<b>Trailblazer.....</b>	<b>vi</b>
<b>About This Guide.....</b>	<b>iv</b>	Overview.....	vi
What This Guide Contains.....	iv	<b>In Depth Look.....</b>	<b>viii</b>
Audience.....	iv	Client.....	viii
Prerequisites.....	iv	Web Service.....	viii
Organization.....	iv	LoanBroker/ESB Components.....	ix
Documentation Conventions.....	iv	Deploying and Testing the Trailblazer.....	xiv
Additional Documentation.....	v	System Requirements.....	xiv
		Configurations.....	xiv
		Running.....	xiv

# About This Guide

## What This Guide Contains

---

The Trailblazer 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 JBoss ESB 4.2 GA.

## Audience

---

This guide is most relevant to engineers who are responsible for using JBoss ESB 4.2 GA installations and want to know how deploy and tes the Trailblazer found under the *Samples*.

## Prerequisites

---

You will need the JBossESB distribution, source or binary to run the trailblazer. You will also need an instance of JBoss Application Server installed with JBossWS and EJB3 support. You can use the App. Server installer from JBoss and install using the **EJB3 Profile**.

To test the email notification of quotes, you will require a mail server or the information from your ISP/company email server.

## Organization

---

This guide contains the following chapters:

- **Chapter 1, Overview:** an overview of the loanbroker trailblazer scenario used in JBossESB.
- **Chapter 2, In Depth Look:** a more detailed look at the various artifacts that make up the trailblazer.
- **Chapter 3, Deploying and Testing the TB:** how to compile, deploy, and test the trailblazer.

## Documentation Conventions

---

The following conventions are used in this guide:

Table 1 Formatting Conventions

Convention	Description
<i>Italic</i>	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.
<b>Bold</b>	Emphasizes items of particular importance.
Code	Text that represents programming code.
<b>Function   Function</b>	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	<p>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:</p> <pre>persistPolicy (Never   OnTimer   OnUpdate   NoMoreOftenThan)</pre>
<b>Note:</b>	A note highlights important supplemental information.
<b>Caution:</b>	A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

### Additional Documentation

---

In addition to this guide, the following guides are available in the JBoss ESB 4.2 GA documentation set:

1. **JBoss ESB 4.2 GA Administration Guide:** How to manage the ESB.
2. **JBoss ESB 4.2 GA Getting Started Guide:** Provides a quick start reference to configuring and using the ESB.
3. **JBoss ESB 4.2 GA Programmers Guide:** How to use JBossESB.
4. **JBoss ESB 4.2 GA Release Notes:** Information on the differences between this release and previous releases.
5. **JBoss ESB 4.2 GA Services Guides:** Various documents related to the services available with the ESB.

### Contacting Us

---

Questions or comments about JBoss ESB 4.2 GA should be directed to our support team.

# Trailblazer

## Overview

---

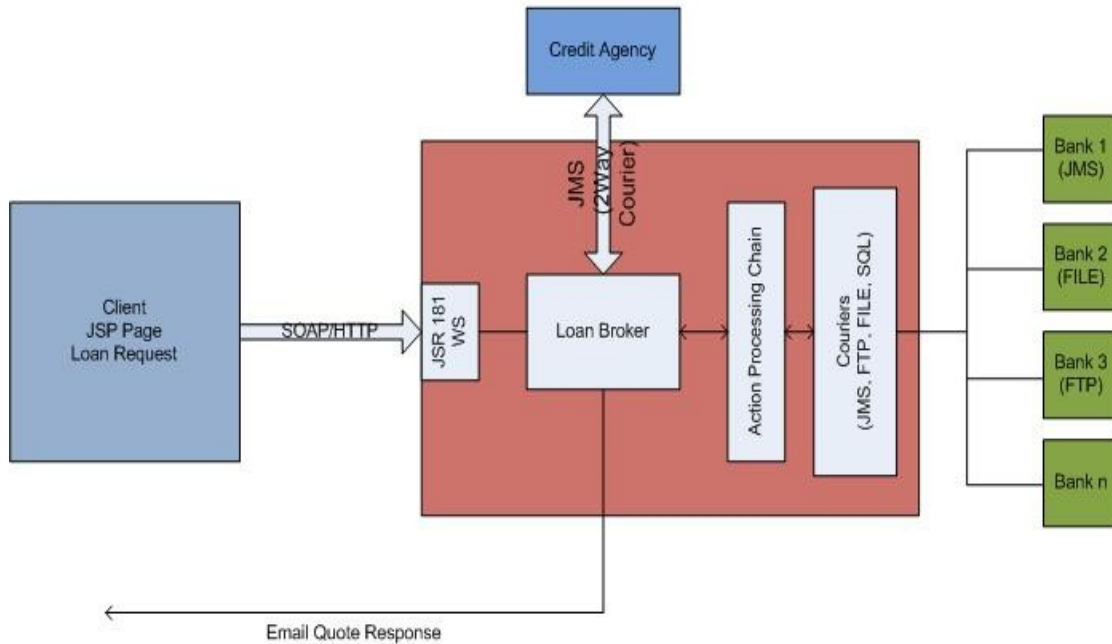
The Trailblazer is meant to show a commonly understood use-case where the JBossESB can be used to solve the integration problem at hand. The TB is loosely based on the Enterprise Applications Integration book (<http://www.eaipatterns.com/>). The scenario is very simple - a user is shopping around for a bank loan with the best terms, rate, etc. A loan broker will act as middle-man between the user and the banks. The LoanBroker will gather all the required information from the user, and the pass it on to each bank. As the quotes are received from the various banks, the LoanBroker will pass those back to the requesting user. This is a common practice in the financial services world today – it's a model used for insurance quotes, mortgage quotes, and so on.

A simple scenario as described above, actually puts forth several integration challenges. Each bank has it's own data feed structure (xml, delimited, positional, etc), it's own communication protocol (file, jms, ftp, etc), and finally the responses from each of these is very unique to each. A LoanBroker acting as the agent for these institutions must be able to accommodate each scenario, without expecting the bank to adjust anything. The bank's provide a service, and have a clearly defined contract in which to carry out that service. It's our job as the LoanBroker developer to ensure we can be as flexible and adaptable as possible to handle a variety of possible communication protocols, data formats and so on.

This is where JBossESB comes in. Traditionally, an organization would create custom code and scripts to manage the end to end integration between the LoanBroker and each bank. (*aka* point-to-point interfaces). This is cumbersome, and messy when it comes to maintenance. Adding new banks, and new protocols is not easy. JBossESB gives us a central framework for developing a solution built around a common set of services which can be applied over and over to each unique bank requirement. Adding a new bank then becomes trivial, and support is a lot simpler when you only need to work on one common codebase.

The diagram below shows the scenario at a high level:

## JBOSS ESB – Trailblazer



\* the diagram above is not using any specific notation or style (some of you might be expecting the EIP symbols).

# In Depth Look

## Client

---

The client is a simple JSP page, which routes the submit to a waiting web service. The Loan Request consists of the typical information you would expect from such a request: A social security number (ssn), some personal information like name, address, and so on, as well as loan specific information – loan amount, etc.

## Web Service

---

The web service, which is responsible for receiving the loan requests is a JSR-181 based annotated web service. An annotated web service lets you take any pojo and expose the methods as being capable of receiving requests. The class looks as follows:

```
package org.JBoss.soa.esb.samples.trailblazer.web;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import org.apache.log4j.Logger;
import org.JBoss.soa.esb.samples.trailblazer.loanbroker.LoanBroker;
/**
 * The Loan broker web service, which will handle a loan request.
 */

@WebService(name = "LoanBrokerWS",
targetNamespace = "http://localhost/trailblazer")
@SOAPBinding(style = SOAPBinding.Style.RPC)

public class LoanBrokerWS
{
    private static Logger logger = Logger.getLogger(LoanBrokerWS.class);
    @WebMethod
    // method name is .NET friendly
    public void RequestLoan(WebCustomer customer) {
        logger.info("WebCustomer received: \n" + customer);
        LoanBroker broker = new LoanBroker();
        broker.processLoanRequest(customer);
    }
}
```

JSR-181 annotated pojo web services are a very easy and powerful way to expose plain old java classes as web services. The JBossESB does not have built in support for web services yet, but since we are working in Java, there is no reason why you cannot combine your own web services with the JBossESB services, which is what was done in the trailblazer. The class above is the server side web service. You still need to provide the client, the JSP in this case the client stubs to communicate with the web service. JBossIDE which has a web service client side generator to create these classes if you are looking for a tool to use for this.



The most important piece in the web service, is the line which invokes the LoanBroker object, and passes a customer object for processing.

## LoanBroker/ESB Components

---

The Loan Broker is a standard java application, which makes use of the services available in the JBossESB to get data to and from the banks, and then finally back to the customer as an email response.

Let's look first at the ESB components required for processing a loan request.

In this release, the Bank bundled include a JMS based bank, and a File based bank. Each has it's own unique data requirements and data formats. These are external entities. In a real production world scenario, these might be internal systems, accessible within your own network, or they may be external providers which you will need to communicate with through some protocol. Needless to say, for this example, we are not focusing on aspects like security, authentication, and other concerns which you would most certainly face. We are focusing solely on the JBossESB components and some sample configurations which you could use to create a similar scenario.

JBossESB has a concept of "Gateway" and "ESB Aware" services. ESB Aware services are able to communicate with the ESB directly using native APIs found in the ESB. These APIs for instance require that you use a Message object. Since the LoanBroker is java based, and has access to the JBossESB APIs, it will be an ESB Aware service. The banks on the other hand, are NON-ESB Aware services. They have no idea, nor should they know anything about the ESB. It is the job of the services in the ESB to facilitate communication to and from the banks, as well as data transformation to/from and so on. These services (the Banks) will interact with the JBossESB through what we call Gateway Services. To read more on the differences between the two, please see the Programmer's Guide.

Let's look at just how you configure the various services in JBossESB. Inside the <TRAILBLAZER\_ROOT>/esb/conf/jbossesb.xml you will see the following deployed services:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<JBossesb
  xmlns="http://anonsvn.labs.JBoss.com/labs/JBossesb/trunk/product/etc/schemas
/xml/JBossesb-1.0.xsd"
  parameterReloadSecs="50">

  <providers>
    <jms-provider name="JBossMQ" connection-factory="ConnectionFactory"
      jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
      jndi-URL="localhost">
      <jms-bus busid="creditAgencyRequest">
        <jms-message-filter dest-type="QUEUE" dest-name="queue/esb-tb-
creditAgencyQueue" selector="function='request' "/>
      </jms-bus>

      <jms-bus busid="bankResponseGateway">
        <jms-message-filter dest-type="QUEUE" dest-name="queue/esb-tb-
bankRequestQueue"/>
      </jms-bus>
```

```

        <jms-bus busid="bankResponseListener">
            <jms-message-filter dest-type="QUEUE" dest-name="queue/esb-tb-
bankResponseQueue"/>
        </jms-bus>
    </jms-provider>
</providers>
<services>
    <service category="trailblazer" name="creditagency" description="Credit
Agency Service">
        <listeners>
            <jms-listener name="trailblazer-jmscreditagency"
                busidref="creditAgencyRequest" maxThreads="1"/>
        </listeners>
        <actions>
            <action
                xclass="org.JBoss.soa.esb.samples.trailblazer.actions.CreditAgencyActions"
                process="processCreditRequest" name="fido"/>
        </action>
        </actions>
    </service>

    <service category="trailblazer-jmsbank" name="jmsbankreplies"
description="Trailblazer Bank Reply Service">
        <listeners>
            <jms-listener name="trailblazer-jmsbank"
                busidref="bankResponseGateway" maxThreads="1"
                is-gateway="true"/>
            <jms-listener name="trailblazer-jmsbankreplies"
                busidref="bankResponseListener" maxThreads="1"/>
        </listeners>
        <actions>
            <action
                class="org.JBoss.soa.esb.samples.trailblazer.actions.BankResponseActions"
                process="processResponseFromJMSBank" name="pepe"/>
        </actions>
    </service>
</services>

</JBossesb>

```

The config above uses a configuration structure which is described in much more detail in Chapter 5 of the JBossESB Programmer's Guide. The config for the TB describes several communication providers, listed in the `<providers>` section, all consisting of JMS in this example, and using JBossMQ as the actual JMS transport. Next, several `<services>` are listed, starting with the *creditagency*, and the various JMS bank services for sending and receiving data from the banks. The banks have their own config files, which must be configured to use and reply on the queues described above. Please see `<TRAILBLAZER_ROOT>/banks/bank.properties`.

The LoanBroker makes use of the services described above, in the following lines of code:

```

public void processLoanRequest(WebCustomer wCustomer) {
    Customer customer = getCustomer(wCustomer);
    //keep the customer in a file someplace for later use, if needed
    CustomerMasterFile.addCustomer(String.valueOf(customer.ssn), customer);
}

```

```

//step 1 - send to credit agency for credit score if available
//uses 2way courier for a response
sendToCreditAgency(customer);

//step 2 - send to JMS Bank
sendToJMSBank(customer);
}

```

The `sendToCreditAgency` is where an interaction with the ESB takes place. Please see the code for more detailed listing. The sections below illustrate the important parts:

```

courier.setReplyToEpr(replyEPR);
//wait for 5 seconds then give up
replyMessage = courier.pickup(5000);

```

We set the courier's *ReplyToEpr* with an EPR we create, then we tell the courier to *pickup* the response for us, waiting a maximum of 5 seconds. For more detailed information on how Couriers and 2WayCourier's work, please see the Programmer's Guide.

The interaction with the Banks uses a simpler, asynchronous API – there is no waiting for a reply from the banks. The bank replies come in on their own queue, and the GatewayService defined for that purpose fires it off to an action class to handle the response. See the listing from the *jbossesb.xml*:

```

<service          category="trailblazer-jmsbank"          name="jmsbankreplies"
description="Trailblazer Bank Reply Service">
  <listeners>
    <jms-listener name="trailblazer-jmsbank"
      busidref="bankResponseGateway" maxThreads="1"
      is-gateway="true"/>
    <jms-listener name="trailblazer-jmsbankreplies"
      busidref="bankResponseListener" maxThreads="1"/>
  </listeners>
  <actions>
    <action
      class="org.JBoss.soa.esb.samples.trailblazer.actions.BankResponseActions"
      process="processResponseFromJMSBank" name="pepe"/>
  </actions>
</service>

```

The important element above is, that the *org.JBoss.soa.esb.samples.trailblazer.actions.BankResponseActions* is the class that is defined as being responsible for handling the bank JMS responses. The property *process="processResponseFromJMSBank"* tells the service which method in this class will actually do the work. Below is a code snippet from this method:

```

public Message processResponseFromJMSBank(Message message) throws Exception
{
  _message = message;
  _logger.debug("message received: \n" + new
String(message.getBody().getContents()));
}

```

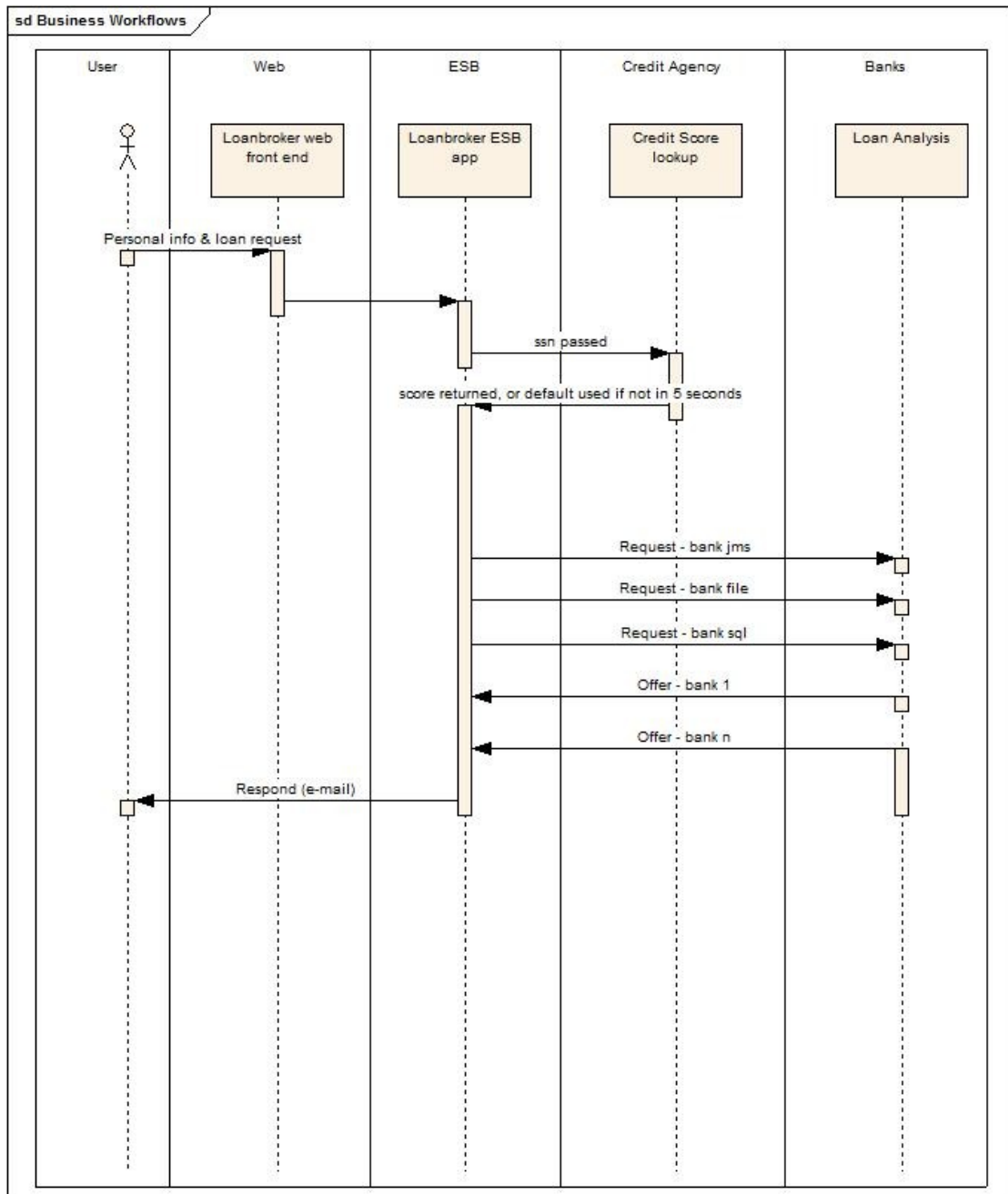
```
//get the response from the bank and set it in the customer
ConfigTree tree = ConfigTree.fromXml(new
Message.getBody().getContents());
String quoteID = tree.getFirstTextChild("quoteId");
String rate = tree.getFirstTextChild("interestRate");
String errorCode = tree.getFirstTextChild("errorCode");
String ssn = tree.getFirstTextChild("customerUID");
String email = tree.getFirstTextChild("customerEmail");

ProcessEmail procEmail = new ProcessEmail(email, quoteID, rate,
errorCode, ssn);
procEmail.sendEmail();

return message;
}
```

The code above retrieves the contents of the payload from the `Message.getBody().getContents()`. Those contents are then used to populate some strings, which are eventually used to fill in the email which goes back to the customer.

The sequence diagram below illustrates the full set of calls that are made in the trailblazers:



## Deploying and Testing the Trailblazer

---

### ***System Requirements***

1. ANT
2. A mail server to send email notifications
3. JBoss AS 4.0.4 or higher with EJB3 and JBossWS support. You can install the latest JBoss AS JEMS installer and select the "EJB3" install config to get the right one installed.
4. JBoss AS 4.0.4 or higher with the JBoss ESB SAR deployment (jbossesb.sar).

### ***Configurations***

1. **jbossesb.sar/jbossesb-properties.xml**: Update the section titled "transports" and specify all of the SMTP mail server settings for your environment.
2. **build.properties**: Update JBoss AS deployment configurations.

### ***Running***

1. Run your JBoss AS - you will need to have the 4.0.4 or higher with the EJB3 support installed. This is required because the TB uses the JSR-181 pojo style web service.
2. From the TB\_ROOT, execute the command to start the ESB: "ant deploy". This should deploy the ESB and WAR files to your JBoss AS server/default.
3. From the TB\_ROOT/banks execute the command to start the Bank service: "ant runJMSBank"
4. In your browser, goto <http://localhost:8080/trailblazer> and submit some quote requests.
5. Check your email for notifications.