Solace JMS Integration with JBoss Application Server EAP 6.2

Document Version 1.5

November 2015

This document is an integration guide for using Solace JMS (starting with version 7.1) as a JMS provider for JBoss Application Server EAP 6.2.

The JBoss Application Server provides a comprehensive framework for application and integration middleware that is compliant with the Java Enterprise Edition computing platform. Solace provides a Java Connector Architecture (JCA) compliant Resource Adapter that may be deployed to the JBoss Application Server providing enterprise applications with connectivity to the Solace message router.

Solace message routers unify many kinds of data movement so companies can efficiently and cost-effectively move all of the information associated with better serving customers and making smarter decisions. The Solace 3560 message router is the highest performance data movement technology available, with the capacity and robustness to support the most demanding enterprise messaging, big data, cloud computing and Internet of Things applications.

Solace Systems®

http://www.solacesystems.com

Table of Contents

1 Overview

This document demonstrates how to integrate Solace Java Message Service (JMS) with the JBoss Application Server EAP 6.2 for production and consumption of JMS messages. The goal of this document is to outline best practices for this integration to enable efficient use of both the application server and Solace JMS.

The target audience of this document is developers using the JBoss Application Server with knowledge of both the JBoss Application Server and JMS in general. As such this document focuses on the technical steps required to achieve the integration.

Note this document provides instructions on configuring and deploying the Solace JCA 1.5 resource adapter in JBoss EAP 6.2 (Enterprise Application Platform). For detailed background on either Solace JMS or the JBoss Application Server refer to the referenced documents below.

This document is divided into the following sections to cover the Solace JMS integration with JBoss Application Server:

- Integrating with JBoss Application Server
- Performance Considerations
- Working with Solace High Availability
- Debugging Tips
- Advanced Topics including:
    - Using SSL Communication
    - Working with XA Transactions
    - Working with Solace Disaster Recovery

1.1 Related Documentation

These documents contain information related to the feature defined in this document

Document ID	Document Title	Document Source
[Solace-Portal]	Solace Developer Portal	http://dev.solacesystems.com
[Solace-JMS-REF]	Solace JMS Messaging API Developer Guide	http://dev.solacesystems.com/docs/solace-jms-api-developer-guide
[Solace-JMS-API]	Solace JMS API Online Reference Documentation	http://dev.solacesystems.com/docs/solace-jms-api-online-reference
[Solace-FG]	Solace Messaging Platform – Feature Guide	http://dev.solacesystems.com/docs/messaging-platform-feature-guide
[Solace-FP]	Solace Messaging Platform – Feature Provisioning	http://dev.solacesystems.com/docs/messaging-platform-feature-provisioning
[Solace-CLI]	Solace Message Router Command Line Interface Reference	http://dev.solacesystems.com/docs/cli-reference
[JBOSS-REF]	JBoss Application Server Information Library	JBoss Application Server 7 Documentation

Document ID	Document Title	Document Source
[JBOSS-SEC]	JBoss Enterprise Application Platform 6.2 Security Guide	JBoss Enterprise Application Platform 6.2 Security Guide
[JCA-1.5]	Java Connector Architecture v1.5	Community Development of Java Technology Specifications (JSR)

Table 1 - Related Documents

2 Why Solace

Solace technology efficiently moves information between all kinds of applications, users and devices, anywhere in the world, over all kinds of networks. Solace makes its state-of-the-art data movement capabilities available via hardware and software “message routers” that can meet the needs of any application or deployment environment. Solace’s unique solution offers unmatched capacity, performance, robustness and TCO so our customers can focus on seizing business opportunities instead of building and maintaining complex data distribution infrastructure.

Superior Performance

Solace’s hardware and software messaging middleware products can cost-effectively meet the performance needs of any application, with feature parity and interoperability that lets companies start small and scale to support higher volume or more demanding requirements over time, and purpose-built appliances that offer 50-100x higher performance than any other technology for customers or applications that require extremely high capacity or low latency.

Robustness

Solace offers high availability (HA) and disaster recovery (DR) without the need for 3rd party products, and fast failover times no other solution can match. Distributing data via dedicated TCP connections ensures an orderly, well-behaved system under load, and patented techniques ensure that the performance of publishers and high-speed consumers is never impacted by slow consumers.

Simple Architecture

Modern enterprises run applications that demand many kinds of data movement such as persistent messaging, web streaming, WAN distribution and cloud-based communications. By supporting all kinds of data movement with a unified platform that can be deployed as a small-footprint software broker or high-capacity rack-mounted appliance, Solace lets architects design an end-to-end infrastructure that’s easy to build applications for, integrate with existing technologies, secure and scale.

Simple Operations

Solace’s solution features a shared administration framework for all kinds of data movement, deployment models and network environments so it’s easy for IT staff to deploy, monitor, manage and upgrade their Solace-based messaging environment.

Cost Savings

Solace reduces expenses with high-capacity hardware, flexible software, and the ability to deploy the right solution for each problem. Solace’s support for many kinds of messaging lets you replace multiple messaging products with just one, built-in HA, DR, WAN and Web functionality eliminate the need for third-party products.

3 Integrating with JBoss Application Server

Solace provides a JCA compliant resource adapter for integrating Java enterprise applications with the Solace JMS message router. There are several options for deploying a Resource Adapter for use by Java enterprise applications including embedded and stand-alone deployment. Solace Systems provides a Resource Adapter Archive (RAR) file for stand-alone deployment.

In order to illustrate JBoss Application Server integration, the following sections will highlight the required JBoss configuration changes and provide sample code for sending and receiving messages using Enterprise Java Beans. The full source code for the ConsumerMDB and ProducerSB Java code can be found in the Section **Appendix - Configuration and Java Source Reference.**

The EJB sample consists of two enterprise beans, a Message Driven Bean (ConsumerMDB) and a Session Bean (ProducerSB). The MDB is configured to receive a message on a 'requests' Queue. When the MDB receives a message it then calls a method of the Session Bean to send a reply message to a 'reply' Queue. The EJB sample requires configuration of various JCA administered objects in JBoss to support usage of the Solace resource adapter.

The following steps are required to accomplish the above goals of sending and receiving messages using the Solace JMS message router.

- Step 1 - Configure the Solace message router
- Step 2 – Deploy the Solace Resource Adapter to the JBoss Application Server
- Step 3 – Connect to Solace JMS provider
    - Configure resource adapter
    - Create and configure JMS connection factory
- Step 4 – Receive inbound messages using Solace JMS provider
    - Create and configure Activation specification
- Step 5 – Send outbound messages using Solace JMS provider
    - Create and configure JMS administered object

3.1 Description of Resources Required

The Solace JCA 1.5 resource adapter is provided as a standalone RAR file and is versioned together with a specific release of the Solace JMS API. The JMS API libraries are bundled inside a single resource adapter RAR file for deployment to the JBoss Application Server.

Resource	File Location
Solace JCA 1.5 resource adapter stand-alone RAR file	https://sftp.solacesystems.com/~<customer>/<version>/Topic_Routing/APIs/JMS/Current/<release>//sol-jms-ra-<release>.rar

Table 2 –Solace Resource Adapter Requirements

This integration guide will demonstrate creation of Solace resources and configuration of the JBoss Application Server managed resources. The section below outlines the resources that are created and used in the subsequent sections.

3.1.1 Solace Resource Naming Convention

To illustrate this integration example, all named resources created on the Solace appliance will have the following prefixes:

Resource	Prefix
Non-JNDI resource	**solace**_<resource name>
JNDI names	**JNDI/Sol/**<resource name>

Table 3 – Solace Resource Naming Convention

3.1.2 Solace Resources

The following Solace message router resources are required for the integration sample in this document.

Resource	Value	Description
Solace message router IP:Port	__IP:Port__	The IP address and port of the Solace message router message backbone. This is the address a client will use when connecting to the Solace message router to send and receive message. This document uses a value of __IP:PORT__.
Message VPN	solace_VPN	A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
Client Username	solace_user	The client username.
Client Password	solace_password	Optional client password.
Solace Queue	solace_requests	Solace destination for messages consumed by JEE enterprise application
Solace Queue	solace_replies	Solace destination for messages produced by JEE enterprise application
JNDI Connection Factory	JNDI/Sol/CF	The JNDI Connection factory for controlling Solace JMS connection properties
JNDI Queue Name	JNDI/Sol/Q/requests	The JNDI name of the queue used in the samples
JNDI Queue Name	JNDI/Sol/Q/replies	The JNDI name of the queue used in the samples

Table 4 – Solace Configuration Resources

3.1.3 Application Server Resource Naming Convention

To illustrate this integration example, all JNDI names local to the JBoss application server have the following prefix:

Resource	Prefix
JNDI names	java:/jms/<resource name>

Table 5 – Application Server Resource Naming Convention

3.1.4 Application Server Resources

The following JBoss Application Server resources are required for the integration example in this document.

Resource	Value	Description
Resource Adapter	com.solacesystems.ra	The name of the *Solace JMS Resource Adapter* module as referenced in the JBoss 'resource-adapters:1.1' subsystem.
JCA connection factory	java:/jms/CF	The connection factory resource referenced by EJB code to perform a JNDI lookup of a Solace javax.jms.ConnectionFactory
JCA administered object	java:/jms/Q/requests	The administered object resource referenced by EJB code to perform a JNDI lookup of a Solace javax.jms.Queue
JCA administered object	java:/jms/Q/replies	The administered object resource referenced by EJB code to perform a JNDI lookup of a Solace javax.jms.Queue

Table 6 – JBoss Configuration Resources

3.2 Step 1 –Solace JMS provider Configuration

The following entities on the Solace message router need to be configured at a minimum to enable JMS to send and receive messages within the JBoss Application Server.

- A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
- Client connectivity configurations like usernames and profiles
- Guaranteed messaging endpoints for receiving and sending messages.
- Appropriate JNDI mappings enabling JMS clients to connect to the Solace message router configuration.

For reference, the CLI commands in the following sections are from SolOS version 6.2 but will generally be forward compatible. For more details related to Solace message router CLI see [Solace-CLI]. Wherever possible, default values will be used to minimize the required configuration. The CLI commands listed also assume that the CLI user has a Global Access Level set to Admin. For details on CLI access levels please see [Solace-FG] section "User Authentication and Authorization".

Also note that this configuration can also be easily performed using SolAdmin, Solace's GUI management tool. This is in fact the recommended approach for configuring a Solace message router. This document uses CLI as the reference to remain concise.

3.2.1 Creating a Message VPN

This section outlines how to create a message-VPN called "solace_VPN" on the Solace message router with authentication disabled and 2GB of message spool quota for Guaranteed Messaging. This message-VPN name is required in the JBoss Application Server configuration when connecting to the Solace messaging appliance. In practice appropriate values for authentication, message spool and other message-VPN properties should be chosen depending on the end application's use case.

```
(config)# create message-vpn solace_VPN
(config-msg-vpn)# authentication
(config-msg-vpn-auth)# user-class client
(config-msg-vpn-auth-user-class)# basic auth-type none
(config-msg-vpn-auth-user-class)# exit
(config-msg-vpn-auth)# exit
(config-msg-vpn)# no shutdown
(config-msg-vpn)# exit
(config)#
(config)# message-spool message-vpn solace_VPN
(config-message-spool)# max-spool-usage 2000
(config-message-spool)# exit
(config)#
```

3.2.2 Configuring Client Usernames & Profiles

This section outlines how to update the default client-profile and how to create a client username for connecting to the Solace message router. For the client-profile, it is important to enable guaranteed messaging for JMS messaging and transacted sessions if using transactions.

The chosen client username of “solace_user” will be required by the JBoss Application Server when connecting to the Solace message router.

```
(config)# client-profile default message-vpn solace_VPN
(config-client-profile)# message-spool allow-guaranteed-message-receive
(config-client-profile)# message-spool allow-guaranteed-message-send
(config-client-profile)# message-spool allow-guaranteed-endpoint-create
(config-client-profile)# message-spool allow-transacted-sessions
(config-client-profile)# exit
(config)#
(config)# create client-username solace_user message-vpn solace_VPN
(config-client-username)# acl-profile default
(config-client-username)# client-profile default
(config-client-username)# no shutdown
(config-client-username)# exit
(config)#
```

3.2.3 Setting up Guaranteed Messaging Endpoints

This integration guide shows receiving messages and sending reply messages within the JBoss Application Server using two separate JMS Queues. For illustration purposes, these queues are chosen to be exclusive queues with a message spool quota of 2GB matching quota associated with the message VPN. The queue names chosen are "solace_requests" and "solace_replies".

```
(config)# message-spool message-vpn solace_VPN
(config-message-spool)# create queue solace_requests
(config-message-spool-queue)# access-type exclusive
(config-message-spool-queue)# max-spool-usage 2000
(config-message-spool-queue)# permission all delete
(config-message-spool-queue)# no shutdown
(config-message-spool-queue)# exit
(config-message-spool)# create queue solace_replies
(config-message-spool-queue)# access-type exclusive
(config-message-spool-queue)# max-spool-usage 2000
(config-message-spool-queue)# permission all delete
(config-message-spool-queue)# no shutdown
(config-message-spool-queue)# exit
(config-message-spool)# exit
(config)#
```

3.2.4 Setting up Solace JNDI References

To enable the JMS clients to connect and look up the Queue destination required by JBoss Application Server, there are three JNDI objects required on the Solace message router:

- A connection factory: JNDI/Sol/CF
- A queue destination: JNDI/Sol/Q/requests
- A queue destination: JNDI/Sol/Q/replies

They are configured as follows (Note, ensure that the 'xa' property is set to 'true' for the JNDI connection factory regardless of whether or not connections will be used for transacted messaging. Refer to *Section 7.3 Working with XA Transactions* for further details):

```
(config)# jndi message-vpn solace_VPN
(config-jndi)# create connection-factory JNDI/Sol/CF
(config-jndi-connection-factory)# property-list messaging-properties
(config-jndi-connection-factory-pl)# property default-delivery-mode persistent
(config-jndi-connection-factory-pl)# property xa true
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property direct-transport false
(config-jndi-connection-factory-pl)# property "reconnect-retry-wait" "3000"
(config-jndi-connection-factory-pl)# property "reconnect-retries" "20"
(config-jndi-connection-factory-pl)# property "connect-retries-per-host" "5"
(config-jndi-connection-factory-pl)# property "connect-retries" "1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)#
(config-jndi)# create queue JNDI/Sol/Q/requests
(config-jndi-queue)# property physical-name solace_requests
(config-jndi-queue)# exit
(config-jndi)#
(config-jndi)# create queue JNDI/Sol/Q/replies
(config-jndi-queue)# property physical-name solace_replies
(config-jndi-queue)# exit
(config-jndi)#
(config-jndi)# no shutdown
(config-jndi)# exit
(config)#
```

3.3 Step 2 – Deploying Solace JCA Resource Adapter

Solace provides a JCA compliant Resource Adapter that can be deployed to the JBoss Application Server to allow Enterprise Java Beans to connect to Solace through a standard JCA interface. This integration guide outlines the steps required to deploy the Solace resource adapter (provided as a stand-alone RAR file) to JBoss.

The release of JBoss EAP 6.2 introduced a new Modular Class Loading mechanism which provides fine-grained isolation of Java classes for deployed applications. The following deployment instructions provide the steps to deploy the Solace JCA Resource Adapter as a JBoss Global Module.

3.3.1 Resource Adapter Deployment Steps

The following steps will make the Solace resource adapter available to all enterprise applications (Refer to *Section 3.1 Description of Resources Required* for the file location of the Solace JCA 1.5 Resource Adapter RAR file).

JBoss allows the developer to configure a specific JCA resource adapter to use for an EJB using either JBoss specific Java annotations or through JBoss deployment descriptor files. This configuration example makes the Solace JCA Resource Adapter Module available to all EJB applications by configuring it as a Global Module. Refer to the 'Class Loading in AS7' section of the [JBOSS-REF] documentation for further details on JBoss Class-Loading mechanisms.

Steps to deploy the Solace JCA Resource Adapter:

1. Create a JBoss module directory for the Solace Resource Adapter

```
<JBoss_Home>/modules/com/solacesystems/ra/main
```

2. Copy the Solace JCA 1.5 Resource Adapter RAR file to the module 'main' directory and unzip the contents of the RAR file. Example contents (where 'xx' is the software version packaged with the specific RAR file):

```
-rw-r--r-- 1 root root  284220 Jan  1  2015 commons-lang-2.6.jar
-rw-r--r-- 1 root root   62050 Jan  1  2015 commons-logging-1.1.3.jar
drwxr-xr-x 2 root root    4096 Jan  1  2015 META-INF
-rw-r--r-- 1 root root  222446 Jan  1  2015 sol-common-7.1.0.xx.jar
-rw-r--r-- 1 root root 1106969 Jan  1  2015 sol-jcsmp-7.1.0.xx.jar
-rw-r--r-- 1 root root  297726 Jan  1  2015 sol-jms-7.1.0.xx.jar
-rw-r--r-- 1 root root  173579 Jan  1  2015 sol-jms-ra-7.1.0.xx.jar
-rw-r--r-- 1 root root 1965375 Jan  2 15:31 sol-jms-ra-7.1.0.xx.rar
```

3. In the module 'main' directory, create a '***module.xml***' file that references the JAR libraries in the Solace JCA 1.5 Resource Adapter and specifies other external dependencies. Example (update the string 'xx' with the JAR versions included in the Solace JCA 1.5 Resource Adapter archive):

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.solacesystems.ra" >
    <resources>
        <resource-root path="." />
        <resource-root path="commons-lang-2.6.jar" />
        <resource-root path="commons-logging-1.1.3.jar" />
        <resource-root path="sol-common-7.1.0.xx.jar" />
        <resource-root path="sol-jcsmp-7.1.0.xx.jar" />
        <resource-root path="sol-jms-7.1.0.xx.jar" />
        <resource-root path="sol-jms-ra-7.1.0.xx.jar"/>
    </resources>

    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
        <module name="javax.resource.api"/>
        <module name="javax.jms.api"/>
    </dependencies>
</module>
```

4. Perform one of the following two steps:
    a. **(Option 1)** Update the '***module.xml***' file in the JBoss JTS subsystem to refer to the Solace Resource Adapter module as a dependency (So that the JTS sub-system has access to the classes of the Solace RA for XA Recovery).
        i. Update the JTS module's *module.xml* file which can be found in the following location:

```
<JBoss Home>/modules/system/layers/base/org/jboss/jts/main
```

        ii. Edit the module.xml file and add the com.solacesystems.ra module dependency:

```
:
    <dependencies>
        <module name="org.omg.api" />
        <module name="org.apache.commons.logging"/>
        <module name="org.jboss.logging"/>
        <module name="org.jboss.jts.integration"/>
        <module name="org.jboss.jboss-transaction-spi"/>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
        <module name="javax.resource.api"/>
        <module name="org.hornetq"/>
        <module name="org.jacorb"/>
        <module name="org.jboss.genericjms.provider" />

        <module name="com.solacesystems.ra" />
    </dependencies>
:
```

    b. **(Option 2)** This option avoids the need to modify the JTS subsystem module.xml configuration file. Using this option exposes the Solace JCA RA Java classes to the JTS subsystem for XA Recovery.
        i. Start the JBoss Application Server with the following Java system property setting:
            1. –D**soljmsra.classLoaderOverride=true**
        ii. The above Java system property may be configured in the JBoss application server *$JBOSS_HOME/bin/standalone.conf* file. Refer to [JBOSS-REF] for alternate ways to configure these settings depending on your specific server configuration.

5. Update the JBoss server configuration – '***urn:jboss:domain:ee:1.1***' subsystem to specify the Solace Resource Adapter module as a Global Module:

```
<subsystem xmlns="urn:jboss:domain:ee:1.1">
   <global-modules>
         <module name="com.solacesystems.ra" slot="main"/>
         <module name="org.jboss.common-core" slot="main"/>
   </global-modules>
   <spec-descriptor-property-replacement>true</spec-descriptor-property-replacement>
   <jboss-descriptor-property-replacement>true</jboss-descriptor-property-replacement>
</subsystem>
```

6. Update the JBoss server configuration – '***urn:jboss:domain:ejb3:1.4***' subsystem to specify the Solace Resource Adapter as the default adapter for Message-Driven-Beans:

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.4">
  :
   <mdb>
        <resource-adapter-ref resource-adapter-name="com.solacesystems.ra"/>
        <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
   </mdb>
  :
```

7. Update the JBoss server configuration – '***urn:jboss:domain:resource-adapters:1.1***' subsystem to add the minimum Solace Resource Adapter configuration. Note, the resource adapter archive location is specified as a module path '*com.solacesystems.ra*':

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
     <resource-adapter id="com.solacesystems.ra">
        <module slot="main" id="com.solacesystems.ra"/>
        <transaction-support>XATransaction</transaction-support>
        <config-property name="MessageVPN"/>
        <config-property name="UserName"/>
        <config-property name="Password"/>
        <config-property name="ConnectionURL"/>
        <connection-definitions/>
        <admin-objects/>
     </resource-adapter>
  </resource-adapters>
</subsystem>
```

3.4 Step 3 – Connecting to Solace JMS provider

Connecting to the Solace message router through the Solace JCA Resource Adapter requires configuration of additional resources in JBoss. Two key pieces of information are required including connectivity information to the Solace message router and client authentication data.

The above information is specified across one or more JMS entities depending on your application's JMS message flow (Inbound, Outbound, or both). Configuration of a JMS connection factory is required for outbound message flow, while configuration of the Activation Specification associated with a Message-Driven-Bean - is required for inbound message flow.

The Solace resource adapter includes several custom properties for specifying connectivity and authentication details (Application-Managed credentials) to the Solace message router. Setting these properties at the Resource Adapter level makes the information available to all child JCA entities like Connection Factory, Activation Specification and Administered Objects. The properties can also be overridden at the specific JCA entity level allowing connectivity to multiple Solace message routers.

Steps to configure the Solace JCA Resource Adapter:

1. Update the JBoss server configuration – '***urn:jboss:domain:resource-adapters:1.1***' subsystem and edit the configuration properties of the Solace Resource Adapter. Update the values for the configuration properties 'ConnectionURL', 'UserName', 'Password', and 'MessageVPN':

```
<resource-adapter id="com.solacesystems.ra">
    <module slot="main" id="com.solacesystems.ra"/>
    <transaction-support>XATransaction</transaction-support>
    <config-property name="ConnectionURL">smf://__IP:Port__</config-property>
    <config-property name="MessageVPN">solace_VPN</config-property>
    <config-property name="UserName">UserName</config-property>
    <config-property name="Password">Password</config-property>
    <connection-definitions/>
    <admin-objects/>
</resource-adapter>
```

2. '*ConnectionURL'* property has the format '***smf://__IP:Port__***' (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP).
3. Specify a value for the '*UserName*' property that corresponds to the Solace username (use the value '***solace_user***' for this example).
4. Specify a value for the '*Password'* property that corresponds to the Solace username's password, use the value '***solace_password'***
5. Specify a value for the '*MessageVPN*' property and specify the value corresponding to the Solace message VPN (use the value '***solace_VPN***' for this example).

The following table summarizes the values used for the Resource Adapter configuration properties.

Name	Value	Description
ConnectionURL	smf://__IP:Port__	The connection URL to the Solace message router of the form: smf://__IP:Port__ (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP)
messageVPN	solace_VPN	A Message VPN name (virtual message broker) to scope the integration on the Solace message router.
UserName	solace_user	The client Solace username credentials
Password	default	Client password

Table 7 – Resource Adapter Configuration properties

Steps to configure a JCA connection factory (This example is for non-transacted messaging; refer to *Section 7.3 Working with XA Transactions* for details on configuring XA enabled JCA connection factories):

1. Edit the configuration properties of the Solace Resource Adapter in the 'resource-adapters:1.1' subsystem of the JBoss application server configuration, and add a new ***connection-definition*** entry:

```
<resource-adapter id="com.solacesystems.ra">
  <module slot="main" id="com.solacesystems.ra"/>
  <transaction-support>XATransaction</transaction-support>
  :
  <connection-definitions>
    <connection-definition
       class-name = "com.solacesystems.jms.ra.outbound.ManagedJMSConnectionFactory"
       jndi-name = "java:/jms/myCF"
       enabled="true" pool-name="myCFPool">

       <config-property name="ConnectionFactoryJndiName">JNDI/Sol/CF</config-property>

       <security>
         <application/>
       </security>
       <validation>
         <background-validation>false</background-validation>
       </validation>
  </connection-definition>
</connection-definitions>
:
</resource-adapter>
```

2. Specify the value '**com.solacesystems.jms.ra.outbound.ManagedJMSConnectionFactory**' for the class-name attribute of the *connection-definition*.
3. Edit the local *jndi-name* attribute of the connection factory as referenced by EJB code (for this example use the value '***java:/jms/myCF***')
4. Edit the *connection-definition* configuration property '***ConnectionFactoryJndiName***' (for this example use the value '**JNDI/Sol/CF**')

Note, values for *ConnectionURL, MessageVPN, UserName* and *Password* must also be specified for the JNDI lookup of the connection factory to succeed. In this example, these values are inherited by the connection-definition from the Resource Adapter configuration properties (or alternatively the values may be specified directly as *config-property* entries of the JMS connection-definition).

The following table summarizes the values used for the JMS connection factory configuration properties.

Name	Value	Description
ConnectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.

Table 8 – JMS connection factory Configuration properties

3.4.1 Connecting – Sample code

Sample code for connecting to the Solace message router through a JCA connection factory will be demonstrated in *Section 3.6 Step 5 – Sending outbound messages using Solace JMS provider.* The sample code in this integration guide is triggered by the receipt of a message by a Message-Driven-Bean (MDB) which in turn calls a Session Bean method to publish an outbound reply message.

3.5 Step 4 – Receiving inbound messages using Solace JMS provider

This example uses a Message-Driven-Bean to receive messages from the Solace JMS provider. The bean is bound to an Activation Specification which specifies the Solace JMS destination from which to consume messages as well as the authentication details used to connect to the Solace message router.

3.5.1 Configuration

In JBoss EAP 6.2, Message Driven Bean – Activation Specifications are configured using either EJB 3.0 annotations or through EJB deployment descriptor files. The following example shows the Activation Specification configuration properties available for connecting to a JMS end point on the Solace message router as well as other configuration options.

Note the values for the attributes ('propertyValue') can take the form '${*propertyName*}' where JBoss replaces the values if the *spec-descriptor-property-replacement* and / or *jboss-descriptor-property-replacement* JBoss server configuration properties are set to '*true*' in the '*urn:jboss:domain:ee:1.1*' subsystem (Refer to [JBOSS-REF] for further details).

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName="connectionFactoryJndiName",
            propertyValue="JNDI/Sol/CF"),
        @ActivationConfigProperty(
            propertyName="destinationType",
            propertyValue="javax.jms.Queue"),
        @ActivationConfigProperty(
            propertyName="destination",
            propertyValue="JNDI/Sol/Q/requests"),
        @ActivationConfigProperty(
            propertyName="messageSelector",
            propertyValue=""),
        @ActivationConfigProperty(
            propertyName="subscriptionDurability",
            propertyValue="Durable"),
        @ActivationConfigProperty(
            propertyName="subscriptionName",
            propertyValue=""),
        @ActivationConfigProperty(
            propertyName="clientId",
            propertyValue="ConsumerMDBexample"),
        @ActivationConfigProperty(
            propertyName="batchSize",
            propertyValue="1"),
        @ActivationConfigProperty(
            propertyName="maxPoolSize",
            propertyValue="8"),
        @ActivationConfigProperty(
            propertyName="reconnectAttempts",
            propertyValue="1"),
        @ActivationConfigProperty(
            propertyName="reconnectInterval",
            propertyValue="30")
    }
)
```

Note the following activation configuration properties are mandatory:

- connectionFactoryJndiName

- destinationType
- destination

Steps to define an Activation specification for this Message-Driven-Bean example (Note, the following values are specified in the ***@MessageDriven*** annotation in the code example in *Section 3.5.2 Receiving messages from Solace – Sample Code*):

1. For the '*connectionFactoryJndiName'* property, specify the value '***JNDI/Sol/CF***' (Note, this is the value configured on the Solace message router in *Section 3.2.4 Setting up Solace JNDI References*)
2. For the '*destination'* property, specify the value '***JNDI/Sol/Q/requests***'. (Note, this is the value configured on the Solace message router in *Section 3.2.4 Setting up Solace JNDI References*).
3. For the '*destinationType*' property, specify the value '***javax.jms.Queue***'.

The following table summarizes important values used for the Activation specification configuration properties:

Name	Value	Description
connectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.
destination	JNDI/Sol/Q/requests	The JNDI name of the JMS destination as configured on the Solace message router.
destinationType	javax.jms.Queue	The JMS class name for the desired destination type
batchSize	1	For non-transacted MDBs, the batchSize() is an optimization to read-in 'batchSize' number of messages at a time from the Connection for distribution to MDB threads. Note, for MDB's that are configured to be transacted (XA), the batchSize property is ignored (internally set to '1').
maxPoolSize	8	The maximum size of the MDB Session Pool. One Session services one MDB thread.
reconnectAttempts	1	The number of times to attempt to re-activation of the MDB after activation failure
reconnectInterval	30	The number of seconds between MDB re-activation attempts.

Table 9 – Activation specification Configuration properties

3.5.2 Receiving messages from Solace – Sample Code

The sample code below shows the implementation of a message-driven bean (ConsumerMDB) which listens for JMS messages to arrive on the configured Solace JCA destination (JNDI/Sol/Q/requests - as configured in the Activation specification). Upon receiving a message, the MDB calls the method sendMessage() of the ProducerSB session bean which in turn sends a reply message to a 'reply' Queue destination.

```
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName="connectionFactoryJndiName",
            propertyValue="JNDI/Sol/CF"),
        @ActivationConfigProperty(
            propertyName="destinationType",
            propertyValue="javax.jms.Queue"),
         @ActivationConfigProperty(
            propertyName="destination",
            propertyValue="JNDI/Sol/Q/requests")
    }
)
public class ConsumerMDB implements MessageListener {

    @EJB(beanName = "ProducerSB", beanInterface=Producer.class)
    Producer sb;

    public ConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() +
           " - ConsumerMDB: received message: " + msg);

        try {
               // Send reply message
               sb.sendMessage();

        } catch (JMSException e) {
                     throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

3.6 Step 5 – Sending outbound messages using Solace JMS provider

This example uses an EJB Session Bean to send reply messages using the Solace resource adapter. The bean code requests a JMS Connection from a Solace Managed Connection Factory (myCF) and then sends a reply message to a JMS destination (myReplyQueue) as configured by a JCA administered object.

3.6.1 Configuration

The connection factory used in this example was configured in *Section 3.4 Step 3 – Connecting to Solace JMS provider*. In addition to the connection factory, we must configure a JMS destination for sending reply messages.

Steps to create a JCA administered object (of type Queue)

1. Edit the Solace Resource Adapter definition in the 'resource-adapters:1.1' subsystem of the JBoss application server configuration and add a new ***admin-object*** entry:

```
<resource-adapter id="com.solacesystems.ra">
  <module slot="main" id="com.solacesystems.ra"/>
  <transaction-support>XATransaction</transaction-support>
  :
  <admin-objects>
    <admin-object class-name="com.solacesystems.jms.ra.outbound.QueueProxy"
        jndi-name="java:/jms/myReplyQueue"
        enabled="true" use-java-context="false" pool-name="myReplyQueuePool">
      <config-property name="Destination">JNDI/Sol/Q/replies</config-property>
    </admin-object>
  </admin-objects>
  :
</resource-adapter>
```

2. Specify the value '***com.solacesystems.jms.ra.outbound.QueueProxy***' for the *class-name* attribute of the admin-object.
3. Edit the local JNDI name attribute value of the admin-object as referenced by EJB application code (for this example use the value '***java:/jms/myReplyQueue***')
4. Edit the value for the admin-object configuration property '***Destination***' (for this example use the value '***JNDI/Sol/Q/replies***')

The following table summarizes the values used for the administered object configuration properties:

Name	Value	Description
Destination	JNDI/Sol/Q/replies	The JNDI name of the JMS destination as configured on the Solace message router.

Table 10 – JMS administered object Configuration properties

3.6.2 Sending Messages to Solace – Sample code

The sample code below shows the implementation of a session bean (ProducerSB) that implements a method sendMessage() which sends a JMS message to the Queue destination configured above. The sendMessage() method is called by the ConsumerMDB bean outlined in *Section 3.5 Step 4 – Receiving inbound messages using Solace JMS provider*.

This example uses Java resource injection for the resources 'myCF' and 'myReplyQueue' which are mapped to their respective JCA administered objects using the application deployment descriptor file (refer to the sample application deployment descriptor file following the code example below).

```
@Stateless(name = "ProducerSB")
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)

public class ProducerSB implements Producer, ProducerLocal
{
	@Resource(name = "myCF")
	ConnectionFactory myCF;

	@Resource(name = "myReplyQueue")
	Queue myReplyQueue;

	public ProducerSB() { }

	@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)
	@Override
	public void sendMessage() throws JMSException {

		System.out.println("Sending reply message");
		Connection conn = null;
		Session session = null;
		MessageProducer prod = null;

		try {
			conn = myCF.createConnection();
			session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
			prod = session.createProducer(myReplyQueue);

			ObjectMessage msg = session.createObjectMessage();
			msg.setObject("Hello world!");
			prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
		}
		finally {
			if (prod != null) prod.close();
			if (session != null) session.close();
			if (conn != null) conn.close();
		}
	}
}
```

The sample above requires configuration of JNDI mapped-names to the resource names referenced in the EJB code. The mapping can be defined in the EJB deployment descriptor file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
       version="3.0">
  <display-name>EJBSample</display-name>
  <enterprise-beans>
    <session>
       <ejb-name>ProducerSB</ejb-name>
       <business-local>com.solacesystems.sample.ProducerLocal</business-local>
       <business-remote>com.solacesystems.sample.Producer</business-remote>
       <ejb-class>com.solacesystems.sample.ProducerSB</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Bean</transaction-type>
       <resource-ref>
              <res-ref-name>myReplyQueue</res-ref-name>
              <res-auth>Application</res-auth>
              <mapped-name>java:/jms/myReplyQueue</mapped-name>
       </resource-ref>
       <resource-ref>
              <res-ref-name>myCF</res-ref-name>
              <res-auth>Application</res-auth>
              <mapped-name>java:/jms/myCF</mapped-name>
       </resource-ref>
    </session>
  </enterprise-beans>
</ejb-jar>
```

4 Performance Considerations

The Solace JCA Resource Adapter relies on the JBoss Application Server for managing the pool of JCA connections. Tuning performance for outbound messaging can in part be accomplished by balancing the maximum number of pooled connections available against the number of peak concurrent outbound messaging clients.

For inbound messaging there are different levers that can be tuned for maximum performance in a given environment. The 'batchSize' configuration property of the Solace - Activation Specification (AS) defines the maximum number of messages retrieved at a time from a JMS destination for delivery to a server session. The server session then routes those messages to respective Message Driven Beans. In addition, the 'maxPoolSize' configuration property of the Solace AS defines the maximum number of pooled JMS sessions that can be allocated to MDB threads. Therefore to fine tune performance for inbound messaging, the 'batchSize' and 'maxPoolSize' must be balanced to the rate of incoming messages.

Another consideration is the overhead of performing JNDI lookups to the Solace message router. JBoss implements JNDI caching by default. Resources referenced by a Message Driven Bean through resource injection will trigger an initial JNDI lookup and subsequently use the cached information whenever the MDB instance is reused from the MDB pool. Similarly, Session beans that perform JNDI lookups through a JNDI Context will have that information cached in association with that context. Whenever the Session bean instance is reused from the Session bean pool, any lookups using the same JNDI Context will utilize the JNDI cached information.

Note, in order to use the JNDI caching mechanisms within JBoss you must use JMS through a JCA resource adapter and reference JMS end points in your code through JEE resource injection.

Please refer to [JBOSS-REF] for details on modifying the default behavior of JNDI caching.

5 Working with Solace High Availability (HA)

The [Solace-JMS-REF] section “Establishing Connection and Creating Sessions” provides details on how to enable the Solace JMS connection to automatically reconnect to the standby appliance in the case of a HA failover of a Solace message router. By default Solace JMS connections will reconnect to the standby appliance in the case of an HA failover.

In general the Solace documentation contains the following note regarding reconnection:

> Note: When using HA redundant appliances, a fail-over from one appliance to its mate will typically occur in less than 30 seconds, however, applications should attempt to reconnect for at least five minutes.

In *Section3.2.4 Setting up Solace JNDI References*, the Solace CLI commands correctly configured the required JNDI properties to reasonable values. These commands are repeated here for completeness.

```
config)# jndi message-vpn solace_VPN
(config-jndi)# create connection-factory JNDI/Sol/CF
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property "reconnect-retry-wait" "3000"
(config-jndi-connection-factory-pl)# property "reconnect-retries" "20"
(config-jndi-connection-factory-pl)# property "connect-retries-per-host" "5"
(config-jndi-connection-factory-pl)# property "connect-retries" "1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)# exit
(config)#
```

In addition to configuring the above properties for connection factories, care should be taken to configure connection properties for performing JNDI lookups to the Solace message router. These settings can be configured in the JBoss Application Server globally by setting them at the Solace resource adapter level or within individual JCA administered objects.

To configure JNDI connection properties for JNDI lookups, set the corresponding Solace JMS property values (as a semi-colon separated list of name=value pairs) through the ‘ExtendedProps’ configuration property of the Solace resource adapter or JCA administered objects.

JMS Property Name	Example Value
Solace_JMS_JNDI_ConnectRetries	1
Solace_JMS_JNDI_ConnectRetriesPerHost	5
Solace_JMS_JNDI_ConnectTimeout	30000 (milliseconds)
Solace_JMS_JNDI_ReadTimeout	10000 (milliseconds)
Solace_JMS_JNDI_ReconnectRetries	20
Solace_JMS_JNDI_ReconnectRetryWait	3000 (milliseconds)

Table 11 – JMS JNDI Connection Properties

6 Debugging Tips for Solace JMS API Integration

The key component for debugging integration issues with the Solace JMS API is to enable API logging. Enabling API logging from JBoss Application Server is described below.

6.1 How to enable Solace JMS API logging

Logging and the logging levels for Solace Resource Adapter Java packages can be enabled using Log4J style configuration in the JBoss '*urn:jboss:domain:logging:1.3*' subsystem . You can enable logging for one or more of the Solace Resource Adapter Java packages listed below.

Note the trace logs can be found in the JEE server logs directory (example: *$JBOSS_HOME/standalone/**server.log***).

Steps to configure debug tracing for specific Solace API packages:

1. Modify the JBoss server configuration:
    a. In the sub-system '*urn:jboss:domain:logging:1.3*', add entries for one or more of the following Solace Resource Adapter packages (Update the logging level to one of 'FATAL', 'ERROR', 'WARN', 'INFO', 'DEBUG', or 'TRACE').

```
<subsystem xmlns="urn:jboss:domain:logging:1.3">
:
  <logger category="com.solacesystems.jms">
<level name="INFO"/>
  </logger>
  <logger category="com.solacesystems.jndi">
       <level name="INFO"/>
  </logger>
  <logger category="com.solacesystems.jcsmp">
       <level name="INFO"/>
  </logger>
  <logger category="com.solacesystems.common">
       <level name="INFO"/>
  </logger>
:
```

7 Advanced Topics

7.1 Authentication

The integration example illustrated in *Section 2* of this guide uses the authentication information specified in the custom properties of the Solace resource adapter. These authentication properties are used whenever Application Managed authentication is specified for a JCA resource. No matter the authentication mode (Application-Managed or Container-Managed) specified for a resource, the Solace 'MessageVPN' information for a connection is always retrieved from the Solace resource adapter configured properties (or from the configured properties of one of the JCA entities – connection factory, administered object or activation specification).

JBoss supports configuration of Container-Managed authentication for JCA connection factories. The JAAS login module *ConfiguredIdentityLoginModule* can be used to provide EJB Container-supplied sign-on credentials to the Solace message router. Refer to [JBOSS-SEC] for more details on configuring EJB Security.

The Solace message router supports a variety of client authentications schemes as described in [Solace-FG] in the Section "Client Authentication and Authorization". The Solace JCA resource adapter supports a subset of these schemes including 'Basic' authentication and 'SSL Client Certificate' authentication. The default authentication scheme used by the Solace JMS Resource Adapter is AUTHENTICATION_SCHEME_BASIC.

The value of the Solace resource adapter config-property '*extendedProps*' is used to specify an alternate authentication scheme such as 'AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE' . The property value of '*extendedProps*' consists of a semi-colon separated list of Solace JMS property / value pairs (SOLACE_PROPERTY=value). You can specify the required properties for an alternate authentication scheme using this technique. Refer to the document [Solace-JMS-API] for further details on the required JMS properties for configuring SSL client certificate authentication.

Although the authentication scheme AUTHENTICATION_SCHEME_BASIC is the default scheme, that scheme could also have been specified using the 'extendedProps' custom property of the resource adapter.

Configuration property	Value
ExtendedProps	Solace_JMS_Authentication_Scheme=AUTHENTICATION_SCHEME_BASIC

Table 12 – Specifying the Solace Authentication Scheme

7.2 Using SSL Communication

This section outlines how to update the Solace message router and JBoss Application Server configuration to switch the client connection to using secure connections with the Solace message router. For the purposes of illustration, this section uses a server certificate on the Solace message router and basic client authentication. It is possible to configure Solace JMS to use client certificates instead of basic authentication. This is done using configuration steps that are very similar to those outlined in this document. The [Solace-FP] and [Solace-JMS-REF] outline the extra configuration items required to switch from basic authentication to client certificates.

To change a JBoss Application Server from using a plain text connection to a secure connection, first the Solace message router configuration must be updated as outlined in Section 7.2.1 and the Solace JMS configuration within the JBoss Application Server must be updated as outlined in Section 7.2.2.

7.2.1 Configuring the Solace message router

To enable secure connections to the Solace message router, the following configuration must be updated on the Solace message router.

- Server Certificate

- TLS/SSL Service Listen Port
- Enable TLS/SSL over SMF in the Message VPN

The following sections outline how to configure these items.

7.2.1.1 Configure the Server Certificate

Before, starting, here is some background detail on the server certificate required by the Solace message router. This is from the [Solace-FP] section “Setting a Server Certificate”

> To enable the exchange of information through TLS/SSL-encrypted SMF service, you must set the TLS/SSL server certificate file that the Solace message router is to use. This server certificate is presented to a client during the TLS/SSL handshakes. A server certificate used by an appliance must be an x509v3 certificate and it must include a private key. The server certificate and key use an RSA algorithm for private key generation, encryption and decryption, and they both must be encoded with a Privacy Enhanced Mail (PEM) format.
>
> The single server certificate file set for the appliance can have a maximum chain depth of three (that is, the single certificate file can contain up to three certificates in a chain that can be used for the certificate verification).

To configure the server certificate, first copy the server certificate to the Solace message router. For the purposes of this example, assume the server certificate file is named “mycert.pem”.

```
# copy sftp://[<username>@]<ip-addr>/<remote-pathname>/mycert.pem /certs
<username>@<ip-addr>'s password:
#
```

Then set the server certificate for the Solace message router.

```
(config)# ssl server-certificate mycert.pem
(config)#
```

7.2.1.2 Configure TLS/SSL Service Listen Port

By default, the Solace message router accepts secure messaging client connections on port 55443. If this port is acceptable then no further configuration is required and this section can be skipped. If a non-default port is desired, then follow the steps below. Note this configuration change will disrupt service to all clients of the Solace message router and should therefore be performed during a maintenance window when this client disconnection is acceptable. This example assumes that the new port should be 55403.

```
(config)# service smf
(config-service-smf)# shutdown
All SMF and WEB clients will be disconnected.
Do you want to continue (y/n)? y
(config-service-smf)# listen-port 55403 ssl
(config-service-smf)# no shutdown
(config-service-smf)# exit
(config)#
```

7.2.1.3 Enable TLS/SSL within the Message VPN

By default within Solace message VPNs both the plain-text and SSL services are enabled. If the Message VPN defaults remain unchanged, then this section can be skipped. However, if within the current application VPN, this service has been disabled, then for secure communication to succeed it should be enabled. The steps below show how to enable SSL within the SMF service to allow secure client connections from the JBoss Application Server.

```
(config)# message-vpn solace_VPN
(config-msg-vpn)# service smf
(config-msg-vpn-service-smf)# ssl
(config-msg-vpn-service-ssl)# no shutdown
(config-msg-vpn-service-ssl)# exit
(config-msg-vpn-service-smf)# exit
(config-msg-vpn-service)# exit
(config-msg-vpn)# exit
(config)#
```

7.2.2 Configuring the JBoss Application Server

Secure connections to the Solace JMS provider require configuring SSL parameters of JCA objects. Two of these configuration parameters include 'ConnectionURL' and 'ExtendedProps'. Note that the property values for 'ConnectionURL' and 'ExtendedProps' are inherited by JCA connection factory, Activation specification, and administered objects from their parent Resource Adapter. Thus, unless you are connecting to multiple Solace message routers, a best practice is to configure values for 'ConnectionURL' and 'ExtendedProps' in the Solace Resource Adapter, otherwise the SSL related changes should be duplicated across configuration properties for all of the JMS administered objects you want to secure.

The required SSL parameters include modifications to the URL scheme of 'ConnectionURL' (from 'smf' to 'smfs'), and setting additional SSL attributes through the configuration property 'ExtendedProps'. The following sections describe the required changes in more detail.

7.2.2.1 Updating the JMS provider URL (ConnectionURL)

In order to signal to the Solace JMS API that the connection should be a secure connection, the protocol must be updated in the URI scheme. The Solace JMS API has a URI format as follows:

```
<URI Scheme>://[username]:[password]@<IP address>[:port]
```

Recall from Section 3.3, originally, the "ConnectionURL" was as follows:

```
smf://___IP:PORT___
```

This specified a URI scheme of "smf" which is the plaint-text method of communicating with the Solace message router. This should be updated to "smfs" to switch to secure communication giving you the following configuration:

```
smfs://___IP:PORT___
```

Steps to update the ConnectionURL configuration property of a Solace JMS Resource Adapter:

1. Update the JBoss server configuration – '***urn:jboss:domain:resource-adapters:1.1***' subsystem and edit the configuration properties of the Solace Resource Adapter. Update the values for the configuration properties 'ConnectionURL':

```
<resource-adapter id="com.solacesystems.ra">
    <module slot="main" id="com.solacesystems.ra"/>
    <transaction-support>XATransaction</transaction-support>

    <config-property name="ConnectionURL">smfs://__IP:Port__</config-property>

    <config-property name="MessageVPN">solace_VPN</config-property>
    <config-property name="UserName">solace_user</config-property>
    <config-property name="Password">Password</config-property>
    <connection-definitions/>
    <admin-objects/>
</resource-adapter>
```

2. '*ConnectionURL'* property has the format '***smfs://__IP:Port__***' (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP and SMF SSL Port #, note the default SSL Port is '55443).

7.2.2.2 Specifying other SSL Related Configuration

The Solace JMS API must be able to validate the server certificate of the Solace message router in order to establish a secure connection. To do this, the following trust store parameters need to be provided.

First the Solace JMS API must be given a location of a trust store file so that it can verify the credentials of the Solace message router server certificate during connection establishment. This parameter takes a URL or Path to the trust store file.

A value for the parameter 'Solace_JMS_SSL_TrustStore' can be set by modifying the Solace JCA Resource Adapter configuration property '*ExtendedProps*'. The configuration property value for '*ExtendedProps*' is comprised of a semi-colon separated list of Solace JMS parameters:

```
Solace_JMS_SSL_TrustStore=___Path_or_URL___
```

A trust store password may also be specified. This password allows the Solace JMS API to validate the integrity of the contents of the trust store. This is done through the Solace JMS parameter *'Solace_JMS_SSL_TrustStorePassword'*.

```
Solace_JMS_SSL_TrustStorePassword=___Password___
```

There are multiple formats for the trust store file. By default Solace JMS assumes a format of Java Key Store (JKS). So if the trust store file follows the JKS format then this parameter may be omitted. Solace JMS supports two formats for the trust store: "jks" for Java Key Store or "pkcs12". Setting the trust store format is done through the parameter *'Solace_JMS_SSL_TrustStoreFormat'*:

```
Solace_JMS_SSL_TrustStoreFormat=jks
```

In a similar fashion, the authentication scheme used to connect to Solace may be specified using the parameter '*Solace_JMS_Authentication_Scheme'* (Please refer to the document [Solace-JMS-REF] for full list of supported extended parameters):

- AUTHENTICATION_SCHEME_BASIC
- AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE

The integration examples in this guide use basic authentication (the default authentication scheme):

```
Solace_JMS_Authentication_Scheme=AUTHENTICATION_SCHEME_BASIC
```

The following example allows SSL connectivity for connections made through a Solace JCA managed connection factory.

Steps to update the '*ExtendedProps*' configuration property of JMS connection factory:

1. Edit the configuration properties of the Solace Resource Adapter in the '***urn:jboss:domain:resource-adapters:1.1***' subsystem of the JBoss application server configuration and add or update a 'config-property' entry for the configuration property *'ExtendedProps'* for a specific JMS connection factory:

```
<resource-adapter id="com.solacesystems.ra">
  :
  <connection-definitions>
    <connection-definition
       class-name = "com.solacesystems.jms.ra.outbound.ManagedJMSConnectionFactory"
       jndi-name = "java:/jms/myCF"
       enabled="true" pool-name="myCFPool">
       <config-property name="ConnectionFactoryJndiName">JNDI/Sol/CF</config-property>
       <config-property name="ExtendedProps">
Solace_JMS_SSL_TrustStore=___Path_or_URL___;Solace_JMS_SSL_TrustStorePassword=___Password
___;Solace_JMS_SSL_TrustStoreFormat=jks
       </config-property>
       <security>
         <application/>
       </security>
       <validation>
         <background-validation>false</background-validation>
       </validation>
    </connection-definition>
  </connection-definitions>
</resource-adapter>
```

2. Specify and / or supplement the value for the '*ExtendedProps*' configuration property to: '*Solace_JMS_SSL_TrustStore=___Path_or_URL___;Solace_JMS_SSL_TrustStorePassword=___Password_ __;Solace_JMS_SSL_TrustStoreFormat=jks'* (Update the values '__Path_or_URL__' and '__Password__' accordingly)

7.3 Working with XA Transactions

This section demonstrates how to configure the Solace message router to support the XA transaction processing capabilities of the Solace JCA Resource Adapter. In addition, code examples are provided showing JMS message consumption and production over XA transactions using both Container-Managed-Transactions (CMT) and Bean-Managed-Transaction (BMT) configuration.

XA transactions are supported in the general-availability release of SolOS version 7.1. The Solace JCA Resource Adapter, by default, provides XA Transaction support for Enterprise Java Beans.

In addition to the standard XA Recovery functionality provided through the Solace JCA Resource Adapter, SolOS version 7.1 provides XA transaction administration facilities in the event that customers must perform manual failure recovery. Refer to the document [Solace-JMS-REF] for full details on administering and configuring XA Transaction support on the Solace Message Router.

7.3.1 Enabling XA Support for JMS Connection Factories – Solace Message Router

To enable XA transaction support for specific JMS connection factories the customer must configure XA support for the respective JNDI connection factory on the Solace Message Router:

```
(config)# jndi message-vpn solace_VPN
(config-jndi)# connection-factory JNDI/Sol/CF
(config-jndi-connection-factory)# property-list messaging-properties
(config-jndi-connection-factory-pl)# property xa true
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)#
```

7.3.2 Enabling XA Recovery Support for JCA Connection Factories – JBoss

To enable XA Recovery for specific JCA connection factories in JBoss the customer must update the connection factory definition with the Solace message router sign-on credentials that will be used by the JTS subsystem during XA-recovery. In addition the customer may also want to modify XA connection pool settings from default values.

Steps to enable XA-recovery for a JCA connection factory:

1. Edit the configuration properties of the Solace Resource Adapter in the '***urn:jboss:domain:resource-adapters:1.1***' subsystem of the JBoss application server configuration and add or update the '*recovery*' sign-on credentials. The *user-name* and *password* values may be specified using replaceable JBoss property names (Example: '*${solace.recovery.user}*'). Note the property 'solace.recovery.user' may be defined in the JBoss Server Bootstrap Script Configuration file (Example: <*JBOSS_HOME*>*/bin/standalone.conf* by setting JAVA_OPTS="$JAVA_OPTS –Dsolace.recovery.user=solace_user"):

```
<connection-definitions>
  <connection-definition
     class-name = "com.solacesystems.jms.ra.outbound.ManagedJMSConnectionFactory"
     jndi-name = "java:/jms/myCF"
     enabled="true" pool-name="myCFXAPool">
     <config-property name="ConnectionFactoryJndiName">JNDI/Sol/CF</config-property>
     <xa-pool>
       <min-pool-size>0</min-pool-size>
       <max-pool-size>10</max-pool-size>
       <prefill>false</prefill>
       <use-strict-min>false</use-strict-min>
       <flush-strategy>FailingConnectionOnly</flush-strategy>
       <pad-xid>false</pad-xid>
       <wrap-xa-resource>true</wrap-xa-resource>
     </xa-pool>
     </config-property>
     <security>
       <application/>
     </security>
     <validation>
       <background-validation>false</background-validation>
     </validation>
     <recovery>
       <recover-credential>
         <user-name>${solace.recovery.user}</user-name>
         <password>${solace.recovery.password}</password>
       </recover-credential>
     </recovery>
  </connection-definition>
```

7.3.3 XA Transactions – Sample Code

The following examples demonstrate how to receive and send messages using XA transactions. Examples are given for both Bean-Managed and Container-Managed Transactions (BMT and CMT respectively).

7.3.3.1 Receiving messages from Solace over XA transaction – CMT Sample Code

The following code is similar to the example from *Section 2* but specifies Container-Managed XA Transaction support for inbound messages. In this example, the Message-Driven-Bean (MDB) - 'XAConsumerMDB' is configured such that the EJB container will provision and start an XA transaction prior to calling the onMessage() method and finalize or rollback the transaction when onMessage() exits (Rollback typically occurs when an unchecked exception is caught by the Container).

```
@TransactionManagement(value = TransactionManagementType.CONTAINER)
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)

public class XAConsumerMDB implements MessageListener {

    @EJB(beanName = "XAProducerSB", beanInterface=Producer.class)
    Producer sb;

    public XAConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() + " - XAConsumerMDB: received
message: " + msg);

        try {
               // Send reply message
               sb.sendMessage();

        } catch (JMSException e) {
                     throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

7.3.3.2 Sending Messages to Solace over XA Transaction – CMT Sample Code

The following code is similar to the EJB example from *Section 2* but configures Container-Managed XA Transaction support for outbound messaging. In this example, the Session Bean 'XAProducerSB' method 'SendMessage()' requires that the caller have an existing XA Transaction context. In this example, the 'SendMessage()' method is called from the MDB - 'XAConsumerMDB' in the above example where the EJB container has created an XA Transaction context for the inbound message. When the method sendMessage() completes the EJB container will either finalize the XA transaction or perform a rollback operation.

```
@Stateless(name = "XAProducerSB")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class XAProducerSB implements Producer, ProducerLocal
{
	@Resource(name = "myCF")
	ConnectionFactory myCF;

	@Resource(name = "myReplyQueue")
	Queue myReplyQueue;

	public XAProducerSB() { }

	@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
	@Override
	public void sendMessage() throws JMSException {

		System.out.println("Sending reply message");
		Connection conn = null;
		Session session = null;
		MessageProducer prod = null;

		try {
			conn = myCF.createConnection();
			session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
			prod = session.createProducer(myReplyQueue);

			ObjectMessage msg = session.createObjectMessage();
			msg.setObject("Hello world!");
			prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
		}
		finally {
			if (prod != null) prod.close();
			if (session != null) session.close();
			if (conn != null) conn.close();
		}
	}
}
```

7.3.3.3 Sending Messages to Solace over XA Transaction – BMT Sample Code

EJB code can use the *UserTransaction* interface (Bean-Managed) to provision and control the lifecycle of an XA transaction. The EJB container will not provision XA transactions when the EJB class's 'TransactionManagement' type is designated as 'BEAN' managed. In the following example, the session Bean 'XAProducerBMTSB' starts a new XA Transaction and performs an explicit 'commit()' operation after successfully sending the message. If a runtime error is detected, then an explicit 'rollback()' operation is executed. If the rollback operation fails, then the EJB code throws an EJBException() to allow the EJB container to handle the error.

```
@Stateless(name = "XAProducerBMTSB")
@TransactionManagement(value=TransactionManagementType.BEAN)
public class XAProducerBMTSB implements Producer, ProducerLocal
{
        @Resource(name = "myCF")
        ConnectionFactory myCF;
        @Resource(name = "myReplyQueue")
        Queue myReplyQueue;
        @Resource
        SessionContext sessionContext;

        public XAProducerBMTSB () { }
        @Override
        public void sendMessage() throws JMSException {
                System.out.println("Sending reply message");
                Connection conn = null;
                Session session = null;
                MessageProducer prod = null;
                UserTransaction ux = sessionContext.getUserTransaction();
                try {
                        ux.begin();
                        conn = myCF.createConnection();
                        session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
                        prod = session.createProducer(myReplyQueue);
                        ObjectMessage msg = session.createObjectMessage();
                        msg.setObject("Hello world!");
                        prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
                        ux.commit();
                } catch (Exception e) {
                        e.printStackTrace();
                        try {
                           ux.rollback();
                        } catch (Exception ex) {
                           throw new EJBException(
                            "rollback failed: " + ex.getMessage(), ex);
                        }
                }
                finally {
                        if (prod != null) prod.close();
                        if (session != null) session.close();
                        if (conn != null) conn.close();
                }
        }
}
```

7.4 Working with the Solace Disaster Recovery Solution

The [Solace- FG] section “Data Center Replication” contains a sub-section on “Application Implementation” which details items that need to be considered when working with Solace’s Data Center Replication feature. This integration guide will show how the following items required to have a JBoss Application Server successfully connect to a backup data center using the Solace Data Center Replication feature.

- Configuring a Host List within the JBoss Application Server
- Configuring JMS Reconnection Properties within Solace JNDI
- Configuring Message Driven Bean Re-activation in the Event of Activation Failures
- Disaster Recovery Behavior Notes

7.4.1 Configuring a Host List within the JBoss Application Server

As described in [Solace-FG], the host list provides the address of the backup data center. This is configured within the JBoss Application Server through the `ConnectionURL` configuration property value (of a respective JCA entity) as follows:

```
smf://__IP_active_site:PORT__,smf://__IP_standby_site:PORT__
```

The active site and standby site addresses are provided as a comma-separated list of ‘Connection URIs’. When connecting, the Solace JMS connection will first try the active site and if it is unable to successfully connect to the active site, then it will try the standby site. This is discussed in much more detail in the referenced Solace documentation

7.4.2 Configuring reasonable JMS Reconnection Properties within Solace JNDI

In order to enable applications to successfully reconnect to the standby site in the event of a data center failure, it is required that the Solace JMS connection be configured to attempt connection reconnection for a sufficiently long time to enable the manual switch-over to occur. This time is application specific depending on individual disaster recovery procedures and can range from minutes to hours depending on the application. In general it is best to tune the reconnection by changing the “reconnect retries” parameter within the Solace JNDI to a value large enough to cover the maximum time to detect and execute a disaster recovery switch over. If this time is unknown, it is also possible to use a value of “-1” to force the Solace JMS API to reconnect indefinitely.

The reconnect retries is tuned in the Solace message router CLI as follows:

```
config)# jndi message-vpn solace_VPN
(config-jndi)# connection-factory JNDI/Sol/CF
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property "reconnect-retries" "-1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)# exit
(config)#
```

7.4.3 Configuring Message Driven Bean Reactivation in the Event of Activation Failures

If a message driven bean is de-activated during a replication failover, the bean may be successfully re-activated to the replication site if the reconnection properties of the bean's Activation Specification are properly configured. The default reconnection properties of the Activation specification are configured to not re-activate the bean upon de-activation.

To enable JBoss to attempt to re-activate the de-activated MDB, configure the reconnection configuration properties of the Activation specification:

Configuration property	Default Value	Description
reconnectAttempts	0	The number of times to attempt to re-activate an MDB after the MDB has failed to activate
reconnectInterval	10	The time interval in seconds to wait between attempts to re-activate the MDB

Table 13 – MDB Reconnection Settings – J2C AS Configuration properties

7.4.4 Disaster Recovery Behavior Notes

When a disaster recovery switch-over occurs, the Solace JMS API must establish a new connection to the Solace message routers in the standby data center. Because this is a new connection there are some special considerations worth noting. The [Solace-FG] contains the following notes:

> Java and JMS APIs
>
> For client applications using the Java or JMS APIs, any sessions on which the clients have published Guaranteed messages will be destroyed after the switch-over. To indicate the disconnect and loss of publisher flow:
>
> - The Java API will generate an exception from the `JCSMPStreamingPublishCorrelatingEventHandler.handleErrorEx()` that contains a subcode of `JCSMPErrorResponseSubcodeEx.UNKNOWN_FLOW_NAME`.
> - The JMS API will generate an exception from the `javax.jms.ExceptionListener` that contains the error code `SolJMSErrorCodes.EC_UNKNOWN_FLOW_NAME_ERROR`.
>
> Upon receiving these exceptions the client application will know to create a new session.
>
> After a new session is established, the client application can republish any Guaranteed messages that had been sent but not acked on the previous session, as these message might not have been persisted and replicated.
>
> To avoid out-of-order messages, the application must maintain an unacked list that is added to before message publish and removed from on receiving an ack from the appliance. If a connection is re-established to a different host in the hostlist, the unacked list must be resent before any new messages are published.
>
> Note: When sending persistent messages using the JMS API, a producer's send message will not return until an acknowledgment is received from the appliance. Once received, it is safe to remove messages from the unacked list.
>
> Alternatively, if the application has a way of determining the last replicated message—perhaps by reading from a last value queue—then the application can use that to determine where to start publishing.

For integration with JBoss, it's important to consider this interaction in the context of a Message Driven Bean and Session Bean.

7.4.4.1 Receiving Messages in a Message Driven Bean

There is no special processing required during a disaster recovery switch-over specifically for applications receiving messages. After successfully reconnecting to the standby site, it is possible that the application will receive some duplicate messages. The application should apply normal duplicate detection handling for these messages.

7.4.4.2 Sending Messages from a Session Bean

For JBoss applications that are sending messages, there is nothing specifically required to reconnect the Solace JMS connection. However, any messages that were in the process of being sent will receive an error from the Solace Resource Adapter. These messages must be retransmitted as possibly duplicated. The application should catch and handle any of the following exceptions:

- javax.resource.spi.SecurityException
- javax.resource.ResourceException or one of its subclasses
- javax.jms.JMSException

8 Appendix - Configuration and Java Source Reference

8.1 ProducerSB.java (Non-Transacted)

```
@Stateless(name = "ProducerSB")
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)

public class ProducerSB implements Producer, ProducerLocal
{
        @Resource(name = "myCF")
        ConnectionFactory myCF;

        @Resource(name = "myReplyQueue")
        Queue myReplyQueue;

        public ProducerSB() { }

        @TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)
        @Override
        public void sendMessage() throws JMSException {

                System.out.println("Sending reply message");
                Connection conn = null;
                Session session = null;
                MessageProducer prod = null;

                try {
                        conn = myCF.createConnection();
                        session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
                        prod = session.createProducer(myReplyQueue);

                        ObjectMessage msg = session.createObjectMessage();
                        msg.setObject("Hello world!");
                        prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
                }
                finally {
                        if (prod != null) prod.close();
                        if (session != null) session.close();
                        if (conn != null) conn.close();
                }
        }
}
```

8.2 XAProducerSB.java (XA Transacted / CMT)

```
@Stateless(name = "XAProducerSB")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class XAProducerSB implements Producer, ProducerLocal
{
	@Resource(name = "myCF")
	ConnectionFactory myCF;

	@Resource(name = "myReplyQueue")
	Queue myReplyQueue;

	public XAProducerSB() { }

	@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
	@Override
	public void sendMessage() throws JMSException {

		System.out.println("Sending reply message");
		Connection conn = null;
		Session session = null;
		MessageProducer prod = null;

		try {
			conn = myCF.createConnection();
			session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
			prod = session.createProducer(myReplyQueue);

			ObjectMessage msg = session.createObjectMessage();
			msg.setObject("Hello world!");
			prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
		}
		finally {
			if (prod != null) prod.close();
			if (session != null) session.close();
			if (conn != null) conn.close();
		}
	}
}
```

8.3 XAProducerBMTSB.java (XA Transacted / BMT)

```
@Stateless(name = "XAProducerBMTSB")
@TransactionManagement(value=TransactionManagementType.BEAN)
public class XAProducerBMTSB implements Producer, ProducerLocal
{
          @Resource(name = "myCF")
          ConnectionFactory myCF;
          @Resource(name = "myReplyQueue")
          Queue myReplyQueue;
         @Resource
          SessionContext sessionContext;
          public XAProducerBMTSB () { }
          @Override
          public void sendMessage() throws JMSException {
                    System.out.println("Sending reply message");
                    Connection conn = null;
                    Session session = null;
                    MessageProducer prod = null;
               UserTransaction ux = sessionContext.getUserTransaction();
                    try {
                              ux.begin();
                              conn = myCF.createConnection();
                              session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
                              prod = session.createProducer(myReplyQueue);
                              ObjectMessage msg = session.createObjectMessage();
                              msg.setObject("Hello world!");
                              prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
                              ux.commit();
                    } catch (Exception e) {
                              e.printStackTrace();
                              try {
                                 ux.rollback();
                              } catch (Exception ex) {
                                 throw new EJBException(
                              "rollback failed: " + ex.getMessage(), ex);
                              }
                    }
                    finally {
                              if (prod != null) prod.close();
                              if (session != null) session.close();
                              if (conn != null) conn.close();
                    }
          }
}
```

8.4 ConsumerMDB.java (Non-Transacted)

```
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)

public class ConsumerMDB implements MessageListener {

    @EJB(beanName = "ProducerSB", beanInterface=Producer.class)
    Producer sb;

    public ConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() + " - ConsumerMDB: received
message: " + msg);

        try {
               // Send reply message
               sb.sendMessage();

        } catch (JMSException e) {
                      throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

8.5 XAConsumerMDB.java (XA Transacted / CMT)

```
@TransactionManagement(value = TransactionManagementType.CONTAINER)
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)

public class XAConsumerMDB implements MessageListener {

    @EJB(beanName = "XAProducerSB", beanInterface=Producer.class)
    Producer sb;

    public XAConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() + " - XAConsumerMDB: received
message: " + msg);

        try {
               // Send reply message
               sb.sendMessage();

        } catch (JMSException e) {
                      throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

8.6 ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
       version="3.0">
  <display-name>EJBSample</display-name>
  <enterprise-beans>
    <session>
       <ejb-name>ProducerSB</ejb-name>
       <business-local>com.solacesystems.sample.ProducerLocal</business-local>
       <business-remote>com.solacesystems.sample.Producer</business-remote>
       <ejb-class>com.solacesystems.sample.ProducerSB</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Bean</transaction-type>
       <resource-ref>
              <res-ref-name>myReplyQueue</res-ref-name>
              <res-auth>Application</res-auth>
              <mapped-name>java:/jms/myReplyQueue</mapped-name>
       </resource-ref>
       <resource-ref>
              <res-ref-name>myCF</res-ref-name>
              <res-auth>Application</res-auth>
              <mapped-name>java:/jms/myCF</mapped-name>
       </resource-ref>
    </session>
  </enterprise-beans>
</ejb-jar>
```

9 Appendix - Solace Resource Adapter JCA Configuration Properties

The following tables list Resource Adapter Configuration properties described in this document.

Note: Please refer to [Solace-JMS-REF] for a full list of properties supported by the Solace Resource Adapter.

Resource Adapter Class Name and Configuration Properties

Class name: com.solacesystems.jms.ra.SolJMSRA

Name	Example Value	Description
ConnectionURL	smf://__IP:Port__	The connection URL to the Solace message router of the form: smf://__IP:Port__ (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP)
ExtendedProps	PROPNAME1=VALUE;PROPNAME2=VALUE;…	Semi-colon separated list of Solace-specific extended properties
MessageVPN	<Solace message VPN name>	A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
UserName	<Solace username>	The client Solace username credentials
Password	*****	Client password

Connection Factory Classes

Interface	Implemented by Solace class
javax.jms.ConnectionFactory	com.solacesystems.jms.ra.outbound.ManagedConnectionFactory
javax.jms.QueueConnectionFactory	com.solacesystems.jms.ra.outbound.ManagedQueueConnectionFactory
javax.jms.TopicConnectionFactory	com.solacesystems.jms.ra.outbound.ManagedTopicConnectionFactory

Connection Factory Configuration Properties

Name	Example Value	Description
ClientId		A JMS client ID.
ConnectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.
ConnectionValidationEnabled	True	When set to true, the application server will be notified of any connection error event emitted through a JMS Connection's Exception listener

Activation Specification Configuration Properties

Name	Example Value	Description
batchSize		
clientId	<client_id>	A JMS client ID.
connectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.
destination	JNDI/Sol/Q/requests	The JNDI name of the JMS destination as configured on the Solace message router.
destinationType	javax.jms.Queue	The JMS class name for the desired destination type
messageSelector		JMS message selector string
subscriptionDurability	Durable	JMS subscription durability (defaults to 'Durable' in the Solace 7.1 release)
subscriptionName		JMS subscription name

Administered Object Classes

Interface	Implemented by Solace class
javax.jms.Queue	com.solacesystems.jms.ra.outbound.QueueProxy
javax.jms.Topic	com.solacesystems.jms.ra.outbound.TopicProxy

Administered Object Configuration Properties

Name	Example Value	Description
Destination	JNDI/Sol/Q/requests	The JNDI name of the JMS destination as configured on the Solace message router.