Integration Guide

Tibco Rendezvous

Adaptris
Release 2.x

Table of contents

1 About This Document

The purpose of this document is to provide a brief guide to integrating with Tibco Rendezvous using the Adaptris Integration Framework.

All testing described below was carried out using Tibco Rendezvous 7.5.1 on Mac OS X 10.4.6.

If you are unfamiliar with Tibco Rendezvous it is worth at least skimming document *<install-dir>/doc/pdf/RV_CONCE.PDF* before continuing.

2 Rendezvous post Installation Tasks

Rendezvous is not a pure Java application and any java process connecting to Rendezvous requires access to native libraries. You will need to make some platform-specific configuration changes to ensure that Integration Framework can locate these native libraries.

On Mac OS X this involves adding the following line to *.bash_profile*:

```
DYLD_LIBRARY_PATH=${DYLD_LIBRARY_PATH}:/<tibco-install-dir>/lib
export DYLD_LIBRARY_PATH
```

The main Rendezvous program is a daemon called *rvd (see below)*. This daemon can be started manually by running *<install-dir>/bin/rvd.*

Alternatively, the daemon can be started automatically if it is not already running. To allow this to happen *<tibco-install-dir>/bin* should be added to the PATH environment variable.

3 Rendezvous Basics

3.1 The Rendezvous Daemon - rvd

The main Rendezvous program is a daemon called *rvd* which is expected to run on each computer in the system[1]. In comparison to most JMS brokers *rvd* is a lightweight process. It need not be started explicitly. Provided *rvd* is the PATH applications will start it if it is not already running.

By default *rvd* uses port 7500 for message delivery. An HTTP admin interface runs on port 7580.

[1] Programs may in fact connect to remote *rvd* daemons. This can be configured using the daemon field in *RendezvousClient,* but you may still require a local installation Rendezvous.

rvd is only suitable for messaging over a LAN. A separate daemon program *rvrd* is suitable for programs which are required to communicate over a WAN. Testing to date has been with *rvd* only.

3.2 Messaging domains

Rendezvous supports both point to point and multicast messaging. Multicast 'send subjects' are broadly analogous to JMS topics. 'Point to point' send subjects however cannot be configured and are always generated by Rendezvous. For this reason Rendezvous point to point messaging is primarily suited to replying to request messages. The Integration Framework currently supports multi cast messaging only.

3.3 Message format

The Rendezvous message format consists of a set of fields of self-describing data and is represented in Java by the class *TibrvMsg*. Each field contains one specific data item of a specific data type.

These fields are labelled and may be optionally numerically identified. They can be accessed by name, numeric identifier or using an iterator.

Note that Rendezvous messages are themselves a valid data type, thus allowing messages to be nested. As the Rendezvous message format doesn't easily support collections of a given data type, a nested message is used by *StandardMessageTranslator* to store *AdaptrisMessage* metadata.

Rendezvous messages have a send subject name which describes their destination. These names are hierarchical and use '.' as a delimiter. Consumers can use wild cards in subscriptions.

3.4 Message delivery

By default Rendezvous provides 'reliable' messaging. This means that message receipt is not confirmed by recipients, messages are not persistently stored during processing and messages are not retained by producers for consumers which are currently off line.

Rendezvous can however be configured to provide persistent, durable messaging with message confirmations.

Configuration for both approaches is described below.

3.5 Events and event queues

Rendezvous events drive asynchronous operations such as inbound message delivery. Rendezvous presents events to appropriate call back functions when the event occurs.

For example, create a *TibrvListener* on an event queue for message received events with a certain send subject. NB you are listening for events on an event queue that indicate a message is available, not listening for the message itself.

Events are received by calling dispatch on the event queue; this is generally handled by utility class *TibrvDispatcher*.

The default event queue always exists and is obtained by calling *Tibrv.defaultQueue()* other queues can be created programmatically for e.g. closer control of different types of events. See configuration examples below.

4 Adapter configuration

4.1 Introduction

Classes RendezvousConsumer and RendezvousProducer use implementations of RendezvousClient to send and receive messages. Both classes are connection-less; so a NullConnection is required..

StandardRendezvousClient provides reliable message and CertifiedRendezvousClient provides certified messaging.

Implementations of RendezvousTranslator handle translation from AdaptrisMessage to TibrvMsg and vice versa. Presently StandardMessageTranslator is the only implementation. This is the default and no additional configuration is required.

4.2 Reliable multicast messaging

The Integration Framework can be configured to provide reliable multicast messaging. In this type of deployment messages are not persistent and their successful delivery or otherwise is not confirmed. It is therefore only suitable for applications where message loss is acceptable.

Configuration is minimal if defaults are appropriate. The following producer and consumer will respectively produce / consume from the destination ‘order’.

```
<producer
 xsi:type="java:com.adaptris.core.tibrv.RendezvousProducer">
 <rendezvous-client
 xsi:type="java:com.adaptris.tibrv.StandardRendezvousClient"/>
 <destination
 xsi:type="java:com.adaptris.core.ConfiguredProduceDestination">
 <destination>order</destination>
 </destination>
</producer>
<consumer
 xsi:type="java:com.adaptris.core.tibrv.RendezvousConsumer">
<rendezvous-client
 xsi:type="java:com.adaptris.tibrv.StandardRendezvousClient"/>
<destination
 xsi:type="java:com.adaptris.core.ConfiguredConsumeDestination">
 <destination>order</destination>
 </destination>
</consumer>
```

The service, daemon, network and queue can all be configured in *StandardRendezvousClient* as follows:

```
<rendezvous-client
 xsi:type="java:com.adaptris.tibrv.StandardRendezvousClient">
 <service>service</service>
 <network>network</network>
 <daemon>daemon</daemon>
```

```
 <queue-name>queue</queue-name>
</rendezvous-client>
```

Refer to Tibco Rendezvous doc *<install-dir>/doc/pdf/RV_CONCE.PDF* for further information about these parameters.

4.3 Certified persistent durable multicast messaging

The Integration Framework can also be configured to provide certified persistent durable messaging. In such deployments messages are persistently stored until their delivery is confirmed and are retained for later delivery to consumers which are currently offline.

For this type of messaging the message producer should be configured as follows:

```
<producer
 xsi:type="java:com.adaptris.core.tibrv.RendezvousProducer">
 <destination
 xsi:type="java:com.adaptris.core.ConfiguredProduceDestination">
 <destination>order</destination>
 </destination>
 <rendezvous-client
 xsi:type="java:com.adaptris.tibrv.CertifiedRendezvousClient">
 <unique-name>a unique name</unique-name>
 <ledger-file-name><path-to>/<ledger-file></ledger-file-name>
 <request-old>true</request-old>
 </rendezvous-client>
</producer>
```

If no ledger file name is configured, messages will be certified (i.e redelivery will be attempted until confirmation is received) but not persistent (messages will only be stored in memory while they are waiting acknowledgment) or durable (unacknowledged messages will not be stored between invocations of the Integration Framework). The request old flag has no effect if there is no ledger file name.

The unique name must be unique in the context of all certified components in the system.

Default behaviour is to keep trying to deliver unconfirmed messages indefinitely. A time out period in seconds can be set using *<delivery-time-limit>*.

For debugging and testing purposes it may be useful for a *RendezvousProducer* to receive and log the confirmation messages which indicate that a message has been successfully delivered. This can be achieved by adding the following configuration to *CertifiedRendezvousClient*:

```
<confirmation-subject>subject</confirmation-subject>
```

The subject should be a Tibco Rendezvous internal subject which incorporates the send subject (destination) name. For example the following subject logs all confirmation events:

```
_RV.INFO.RVCM.DELIVERY.CONFIRM.<send subject here>
```

To log all completion events use:

```
_RV.INFO.RVCM.DELIVERY.COMPLETE.<send subject here>
```

The exact nature of events created and received depends on the type of *RendezvousClient* used by the consumer which is receiving the messages. If a *StandardRendezvousClient* is used, no confirmations are produced. NB message delivery is still certified. If a *CertifiedRendezvousClient* is used confirmations are produced.

When configuring a *RendezvousConsumer* with a *CertifiedRendezvousClient*, ledger file name, request old, delivery time limit and confirmation subject are not applicable and will be ignored if set. A unique name must be set. Sample configuration is therefore as follows:

```
<consumer
 xsi:type="java:com.adaptris.core.tibrv.RendezvousConsumer">
 <rendezvous-client
 xsi:type="java:com.adaptris.tibrv.CertifiedRendezvousClient">
 <unique-name>a unique name</unique-name>
 </rendezvous-client>
 <destination
 xsi:type="java:com.adaptris.core.ConfiguredConsumeDestination">
 <destination>order</destination>
 </destination>
</consumer>
```

Service, network, daemon and queue may also be configured as described above.

5 Implementation Notes

The Integration Framework Tibco Rendezvous plug-in does not currently support the following:

- Request-reply messaging.
- Point to point messaging. Rendezvous point to point messaging uses generated rather than configured destinations and is therefore only really suitable for sending replies to requests.
- Messaging over WANs using *rvrd*. Not yet tested.