BEA WebLogic Integration™

TIBCO Rendezvous Control and Event Generator User Guide

Version 8.1 Service Pack 5
Revised: October 2005

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1. TIBCO Rendezvous Control

2. TIBCO Rendezvous Event Generator

3. TIBCO Rendezvous Sample

Index

CHAPTER 1

TIBCO Rendezvous Control

TIBCO® Rendezvous™ (a product from TIBCO: www.tibco.com) enables exchange of data across applications running on distributed platforms. TIBCO Rendezvous (TIBCO RV) Control in WebLogic Integration™ enables seamless connection to, and transfer of data using the Rendezvous daemon. It enables communication via many of the features provided by the TIBCO Rendezvous product, including Certified Message Delivery, Distributed Queue and so on. The sending and receiving applications can be on multiple platforms, as long as the Rendezvous daemon is running on the host machine, or is remotely accessible to the host.

Note: The TIBCO RV control is available in WebLogic Workshop™ only for licensed users of WebLogic Integration.

Disclaimer

Use of the TIBCO RV control and event generator with BEA WebLogic Integration in no manner confers or grants the right to use TIBCO Rendezvous including "dynamic libraries". In order to use such TIBCO products, the user of the TIBCO RV control and event generator must obtain a valid license from TIBCO. See `http://www.tibco.com` for information on how to obtain a licensed copy of Rendezvous.

Topics Included in This Section

Overview: Rendezvous Control

Describes the function of the TIBCO RV control within WebLogic Integration.

Creating and Configuring a New Instance of the TIBCO RV Control
Describes how to create and configure a new TIBCO RV control.

Using Subject in a Message
Describes how to set and retrieve the subject descriptor attributes of the message.

Sending and Receiving Messages
Describes the methods used to send and receive messages.

Setting Dynamic Properties
Describes how to modify the TIBCO RV control properties at run time.

Overview: Rendezvous Control

The TIBCO RV control enables WebLogic Integration business processes to send and receive messages in the Rendezvous environment. In this environment, the messages are conveyed using Rendezvous daemon (`rvd`) and Rendezvous agent (`rva`) transports.

Using the TIBCO RV control, you can send and receive messages in XML, String and TIBCO proprietary Rendezvous Message (`TibrvMsg`) formats. You can specify TIBCO RV control properties while configuring Rendezvous control or dynamically at run time. Following are some of the other features of TIBCO RV control:

- Sending a request message and waiting for a reply
- Sending a reply for a message
- Asynchronous callback facility to confirm delivery or failure of certified messages
- Registration of anticipated listeners

The TIBCO RV control complements the other controls provided in WebLogic Integration, and can be used with other WebLogic Integration business processes. To learn more, see Using Controls in Business Processes in *Using Integration Controls*, which is located at the following URL:

`http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html`.

The TIBCO RV event generator listens on a subject, and publishes the received messages to the WebLogic Integration message broker channels. For more information, see Chapter 2, “TIBCO Rendezvous Event Generator”.

Creating and Configuring a New Instance of the TIBCO RV Control

You can create and configure a new instance of the TIBCO RV control and add it to your business process. This topic includes the following sections:

To Add a New TIBCO RV Control

To add a new TIBCO RV control to WebLogic Integration, perform the following steps:

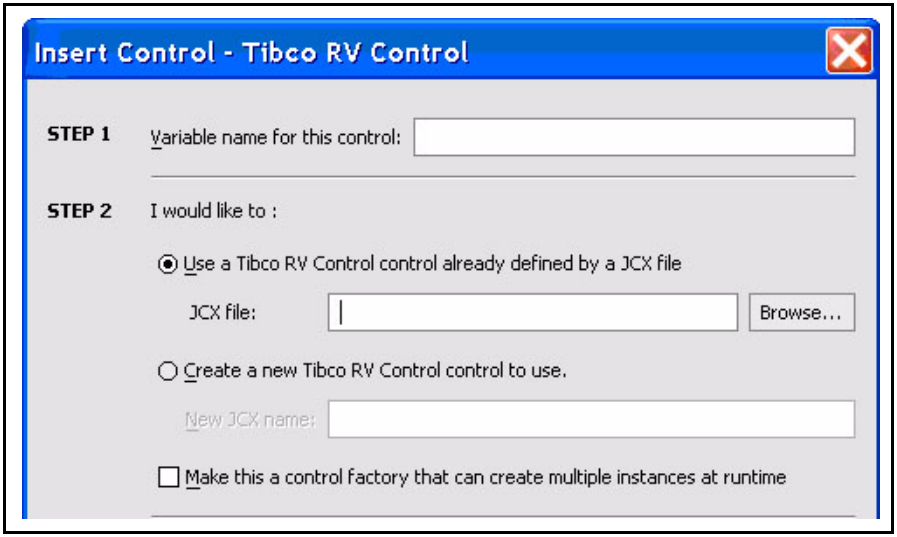
1. Click **Add** on the **Controls** tab to display a list of controls that represent the resources with which your business process can interact.

   Note: If the **Controls** tab is not visible in WebLogic Workshop, choose **View→Windows→Data Palette** from the menu bar.

2. Choose **Integration Controls** to display the list of controls used for integrating applications.

3. Choose **TIBCO RV Control** to display the **Insert Control - TIBCO RV** dialog as shown below.

Figure 1-1 Insert TIBCO RV Control Dialog - Step 1 & 2



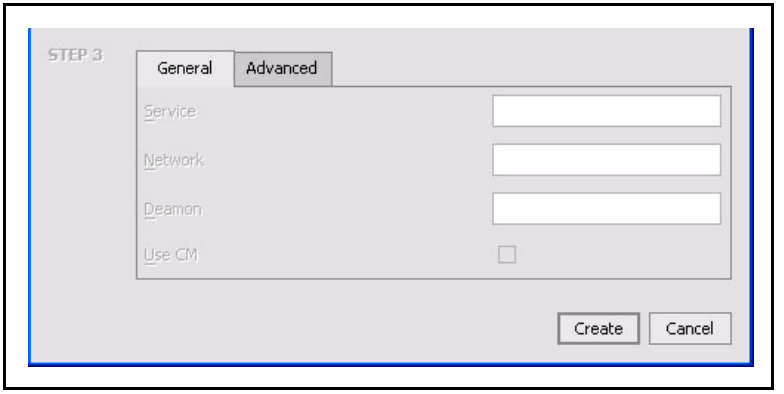
4. In **Step 1**, in the **Variable name for this control** field, enter the name for your TIBCO RV control.
5. In **Step 2**, click the **Create a new TIBCO RV control to use** radio button.

   Note: To use an existing TIBCO RV control, click the **Browse** button to select the JCX file from your system. When you use an existing TIBCO RV control, the properties that were originally selected for the existing control are populated in the JCX file. You cannot modify the properties using the **Insert Control - TIBCO RV Control** dialog. However, you can modify the dynamic properties at run time. For more information, see Setting Dynamic Properties.
6. In the **New JCX name** field, enter the name of the new file.
7. To make this a control factory select the **Make this a control factory that can create multiple instances at runtime** check box, otherwise clear the check box. For more information about control factories, see Control Factories: Managing Collections of Controls.
8. For details on **Step 3**, see To Specify TIBCO RV Control General Settings and To Specify TIBCO RV Control Advanced Settings.
9. Click **Create**.

To Specify TIBCO RV Control General Settings

To specify connection settings for the TIBCO RV control, perform the following tasks in Step 3 of the **Insert Control - TIBCO RV** dialog; the **General Settings** tab, as shown below:

Figure 1-2 Insert TIBCO RV Control Dialog - Step 3: General Settings

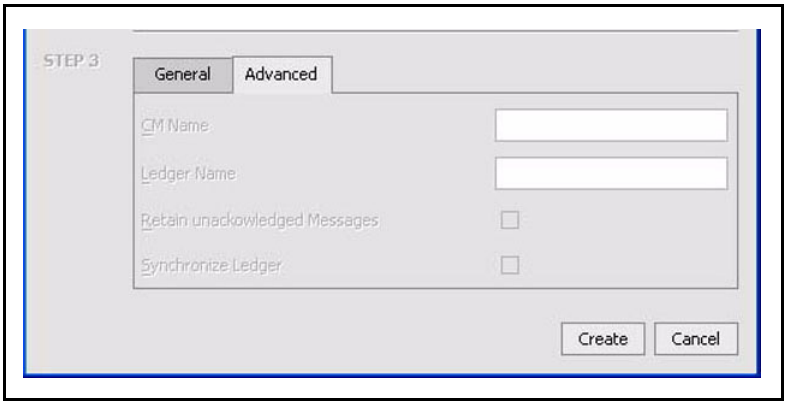


1. In the **Service** field, enter the service name which the TIBCO RV daemon will use to convey the message.
2. In the **Network** field, enter the name of the network with which the TIBCO RV daemon will communicate. If no network is specified, the default network interface will be used.
3. In the **Daemon** field, enter the location where the TIBCO RV daemon is running to establish communication. If the TIBCO RV daemon is running on a different network, specify the **remote_host:port_id** details in the **Daemon** field. For example, **`beaserv1:1589`** where **`beaserv1`** is the remote host name and **`1589`** is the port id.
4. Click **CM** to select the Certified Messaging option.

To Specify TIBCO RV Control Advanced Settings

To specify certified messaging settings for the TIBCO RV control, perform the following tasks in Step 3 of the **Insert Control - TIBCO RV** dialog; the **Advanced Settings** tab, as shown below:

Figure 1-3 Insert TIBCO RV Control Dialog - Step 3: Advanced Settings



1. Click the **Advanced** tab, to display the advanced options for Certified Messaging. This tab is applicable only if you have selected the **CM** radio button in the **General** tab.
2. In the **CM Name** field, provide the CM transport name. The name identifies the CM transport to other CM transports, and is par of the CM label that identifies outbound messages from the CM transport.
3. In the **Ledger Name** field, provide the ledger name with its location. Each CM transport keeps a ledger, in which it records information about every unresolved outbound certified message, every subject for which this CM transport receives (inbound) certified messages, and other cooperating CM transports.
4. Click **Retain Unacknowledged Messages** radio button to store any unacknowledged messages as part of its decentralized architecture.
5. Click **Synchronize Ledger** radio button to perform a synchronized update of the ledger file. Each time the ledger is updated, the call does not return until data is safely written to the storage medium.

The JCX Files for a TIBCO RV Control

When you create a new instance of the TIBCO RV control, you create a new JCX file in your project. The contents of the TIBCO RV control's JCX file depends on the selections made in the **Insert Control - TIBCO RV** dialog.

The two examples in this section depict JCX files created for a certified message and a non-certified message.

Sample JCX File for a TIBCO RV Control Using Certified Messaging

```
package processes;
import com.bea.control.*;
import com.bea.xml.XmlCursor;
import com.bea.control.tibrv.*;
import javax.resource.ResourceException;
import com.bea.xml.XmlObject;
       /*
       * A custom Tibrv control.
       */
       /**
       * @jc:Transport service="7500" //<service name used by the TIBCO
daemon for conveying messages>
              network="beaserv1" //<network with which the TIBCO daemon
will communicate>
              deamon="beasever1:7500 " //<location where the TIBCO daemon
is running>
       * @jc:UseCM  usecm="true" //<indicates certified messaging enabled>
       * @jc:CMTransport cmname="cmname.new" //<the certified message name>
              ledgername="C:\TIBCO\TIBRV\Ledger.txt" //<name and location
of the ledger file>
              requestold="true" //<indicates any unacknowledged messages
will be stored>
              syncledger="true"//<performs synchronized update of the
ledger file>
       */
public interface sendReq extends TibcoRV, com.bea.control.ControlExtension
{
       /*
       * A version number for this JCX. This will be incremented in new
versions of
       * this control to ensure that conversations for instances of earlier
       * versions were invalid.
       */
       static final long serialVersionUID = 1L;
       public void addListenerForCM(String cmName, String subject);
}
```

Sample JCX File for a TIBCO RV Control Without Certified Messaging

```
package processes;
import com.bea.control.*;
import com.bea.xml.XmlCursor;
import com.bea.control.tibrv.*;
import javax.resource.ResourceException;
import com.bea.xml.XmlObject;
       /*
       * A custom Tibrv control.
       */
       /**
       * @jc:Transport service="7500" //<service name used by the TIBCO
daemon for conveying messages>
              network="beaserv1" //<network with which the TIBCO daemon
will communicate>
              deamon="beasever1:7500 " //<location where the TIBCO daemon
is running>
       * @jc:UseCM  usecm="false" //<indicates certified messaging
disabled>
       * @jc:CMTransport cmname=""
              ledgername=""
              requestold="false"
              syncledger="false"
       */
public interface sendReq extends TibcoRV, com.bea.control.ControlExtension
{
       /*
       * A version number for this JCX. This will be incremented in new
versions of
       * this control to ensure that conversations for instances of earlier
       * versions were invalid.
       */
       static final long serialVersionUID = 1L;
}
```

Using Subject in a Message

This section provides details on construction of a subject name. Each message in the TIBCO Rendezvous environment contains a **subject** name. An application creates a message and sends it with a **subject** through the Rendezvous environment. Applications at the other end accept the message by listening on the **subject**.

Subject Name Syntax

Subject name definitions have basic restrictions, for example, its length, structure and usage of special characters. System designers and developers can set the conventions for subject names keeping in mind the following:

- **Structure** — A subject is a string of characters that is divided into elements by the dot (.) character.
- **Length** — The maximum allocated length of a subject (including dot separators) is 255 characters, some of which is reserved for internal use by Rendezvous.
- **Special Characters —**
  - Avoid underscore (_) character at the beginning of the subject name, except if the first element name is `_INBOX` or `_LOCAL`.
  - Avoid the dot (.) character as part of an element as it is the reserved delimiter.
  - Greater-than (>) and Asterisk (*) characters are reservoir wildcard characters.

Caution: The restrictions and conventions are implemented by TIBCO Rendezvous and information in this section is indicative only. Refer TIBCO Rendezvous product documentation for more up-to-date information on restrictions, guidelines and examples.

`http://www.tibco.com`

Sending and Receiving Messages

You can send and receive messages with TIBCO RV control using any one of `sendMessage`, `sendReply` or `sendRequest` functions, and the TIBCO Event Generator, respectively. Messages can be in the form of Rendezvous proprietary data format, string and XML.

Sending Messages

This section provides information on the various functions available for sending messages. To send a message, select a function based on the data type of the message that you want to send. All these functions can send reliable and certified messages, as defined while creating the control. Certified message functions will return sequence numbers while reliable message functions will return zero.

The **sendRequest** function creates a listener that keeps listening for messages to the reply subject and hence, it does not require explicit creation of listeners. The function returns an instance of the `TibrvMsg`, which can be used for sending replies.

The **sendRequest** and **sendReply** functions are often used together as pairs. An example of such an implementation is:

```
replymsg = sendRequest(msg, "send.Subject","reply.Subject",5.0);
sendReply( replyMsg, newMsg);
```

sendMessage ()

Used to send a message via RVDTransport, or a labelled message via CMTransport.

```
public long sendMessage(TibrvMsg msg , String subject, double timeout)
```

msg: the message that needs to be sent
subject: subject of the message
timeout: time limit for delivery of the message

sendMessageAsString ()

Used to send a string message via RVDTransport, or a labelled string message via CMTransport.

```
public long sendMessageAsString(String  msg , String fieldName ,String
subject, double timeout)
```

msg: the string message that needs to be sent
fieldName: name of the TibRV field used to send the payload
subject: subject of the message
timeout: time limit for delivery of the message

sendMessageAsXML ()

Used to send an XML message via RVDTransport, or a labelled XML message via CMTransport.

```
public long sendMessageAsXML(XmlObject  msg ,String fieldName ,String
subject, double timeout)
```

msg: the XML message that needs to be sent
fieldName: name of the TibRV field used to send the payload
subject: subject of the message
timeout: time limit for delivery of the message

sendRequest ()

Used to send a request message via RVDTransport, or a labelled request message via CMTransport and wait for a reply.

```
public TibrvMsg sendRequest(TibrvMsg msg, String sendSubject, double
timeout)
```

msg: the request message that needs to be sent
sendSubject: the send subject of the message
timeout: amount of time to wait for the reply

sendRequestAsString ()

Used to send a request string message via RVDTransport, or a labelled request string message via CMTransport and wait for a reply.

```
public TibrvMsg sendRequestAsString(String msg, String fieldName, String
sendSubject, double timeout)
```

msg: the request string message that needs to be sent
fieldName: name of the TibRV field used to send the payload
sendSubject: subject of the message
timeout: amount of time to wait for the reply

sendRequestAsXML ()

Used to send a request XML message via RVDTransport, or a labelled request message via CMTransport and wait for a reply.

```
public TibrvMsg sendRequestAsXML(XmlObject msg, String fieldName,String
sendSubject, double timeout)
```

msg: the request XML message that needs to be sent
fieldName: name of the TibRV field used to send the payload
sendSubject: subject of the message
timeout: amount of time to wait for the reply

sendReply ()

Used to send a reply via RVDTransport, or a labelled reply via CMTransport.

```
public long sendReply(TibrvMsg replyMsg, TibrvMsg sendMsg, double timeout)
```

replyMsg: the reply message
sendMsg: the request message
timeout: time limit for delivery of the message

sendReplyAsString ()

Used to send a string type reply via RVDTransport, or a labelled string type reply via CMTransport.

```
public long sendReplyAsString(TibrvMsg replyMsg, String sendMsg, String
fieldName, double timeout)
```

replyMsg: the reply string message
sendMsg: the request string message
fieldName: the name of the TibRV field used to send the payload
timeout: time limit for delivery of the message

sendReplyAsXML ()

Used to send an XML type reply via RVDTransport, or a labelled XML type reply via CMTransport.

```
public long sendReplyAsXML(TibrvMsg replyMsg, XmlObject sendMsg, String
fieldName, double timeout)
```

replyMsg: the reply XML message
sendMsg: the request XML message
fieldName: the name of the TibRV field used to send the payload
timeout: time limit for delivery of the message

setStringEncoding ()

Used to set the character encoding for converting between Java Unicode strings and wire format strings.

```
void setStringEncoding(java.lang.String encoding) throws
java.io.UnsupportedEncodingException;
```

encoding: determines encoding

Additional Functions for Certified Messaging

You can include the following two functions when using the CMTransport.

onCMMessageReceipt ()

Used to define a callback method to receive confirmation for message sent. It can only be used with **`sendMessage`** or **`sendReply`** functions. TIBCO RV control subscribes to two confirmation advisories: `_RV.INFO.RVCM.DELIVERY.COMPLETE.>` and `_RV.ERROR.RVCM.DELIVERY.FAILED.>`.

Note: A TIBCO RV control with certified messaging enabled must have an **`onCMReceipt()`** method implemented in the process definition. Without this, a runtime exception will be thrown.

```
public void onCMMessageReceipt(byte[] data);
```

`data`: message data

addListenerForCM ()

Used to pre-register an anticipated listener. When a sending application pre-registers listeners, Rendezvous will store all outbound messages in the sender's ledger. So, when the listener requests certified delivery, it receives the backlogged messages. This function is the same as the `addListener` method in Rendezvous. Refer TIBCO Rendezvous product documentation for more details.

```
void addListenerForCM(String cmName, String subject);
```

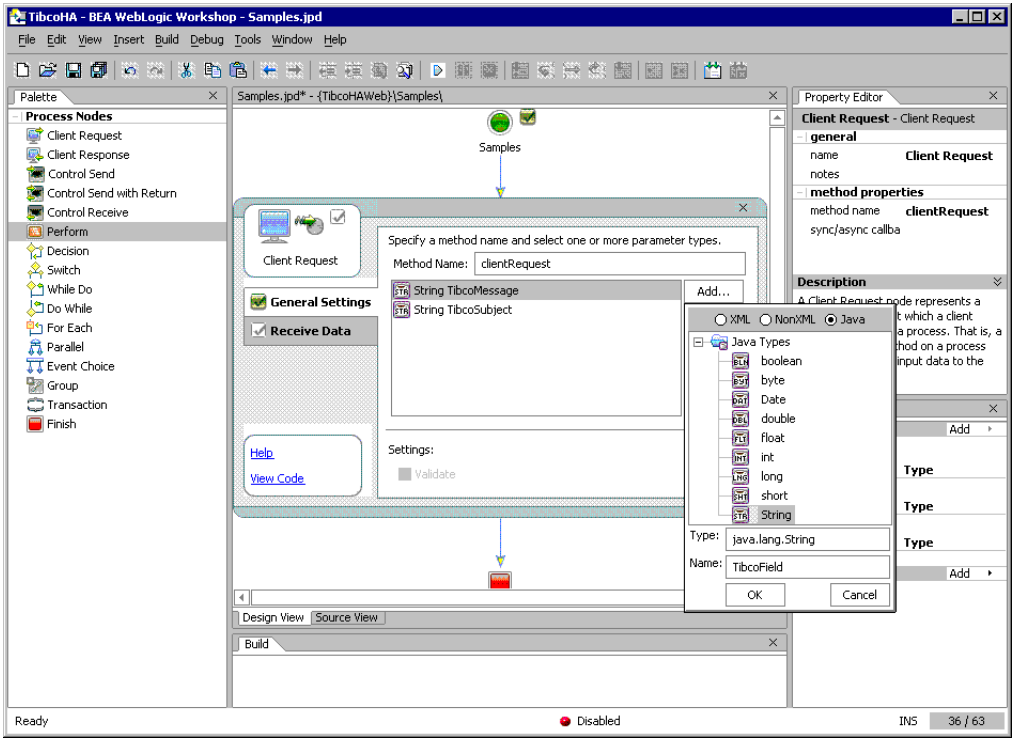
`cmName`: the certified message name
`subject`: subject of the message

Using the sendMessage Function In a Business Process

The following procedure in an example that describes how to add any TIBCO RV control `sendMessage` function to a business process.

1. Open the **Client Request** node, as shown in the following figure.

Figure 1-4 Defining a Method for SendMessage Function



2. In the **General Settings** tab, enter a name for the new method.
3. Click **Add**, and select the Java check box in the pop-up dialog.
4. Select **String** from the **Java Types** list and enter a name for the variable in the **Name** field.
5. Click **OK** to add your selection to the **Client Request** node. This represents the message for the `sendMessage` function.
6. Repeat steps 3 to 5 above to add two more variables to the list. The new variables represent the field name and the subject name of the `sendMessage` function.
7. In the **Receive Data** tab, create a new variable for each parameter that you created in the **General Settings** tab of the **Client Request** node. You must provide variable names for all the parameters. The variable type is pre-defined, based on the parameters to which you are assigning the variable.
8. Close the **Client Request** node.

9. Drag and drop the **Perform** node from the **Process Nodes Palette** and convert the message data from String to **`TibrvMsg`** format. See sample code below:

```
// Generating a Tibrv message from the string data format
public void perform() throws Exception
{
   com.tibco.tibrv.TibrvMsg tibrvMsg = new com.tibco.tibrv.TibrvMsg();
   tibrvMsg.update(TibcoField,TibcoMessage);
}
```

10. Drag and drop the `sendMessage` function from the **Controls** tab in the **Data Palette** into your business process, just below the **Client Request** node.

11. Open the **Send Data** tab of the `sendMessage` function node. From the **Select variables to assign** drop-down list, assign the variables that you created in the **Receive Data** tab of the **Client Request** node, to the corresponding parameter of the `sendMessage` function listed in the **Control Expects** column.

12. Open the **Receive Data** tab of the `sendMessage` function note. From the Select variables to assign drop-down list, create a new variable in which to store the sequence number provided by the `sendMessage` function.

You can use similar steps to send messages using the `sendMessageAsString` or the `sendMessageAsXML` functions. Ignore step 9 above as these functions do not require conversion to TibrvMsg format.

Receiving Messages

To receive messages, use the TIBCO RV Event Generator utility. For details, refer Chapter 2, "TIBCO Rendezvous Event Generator".

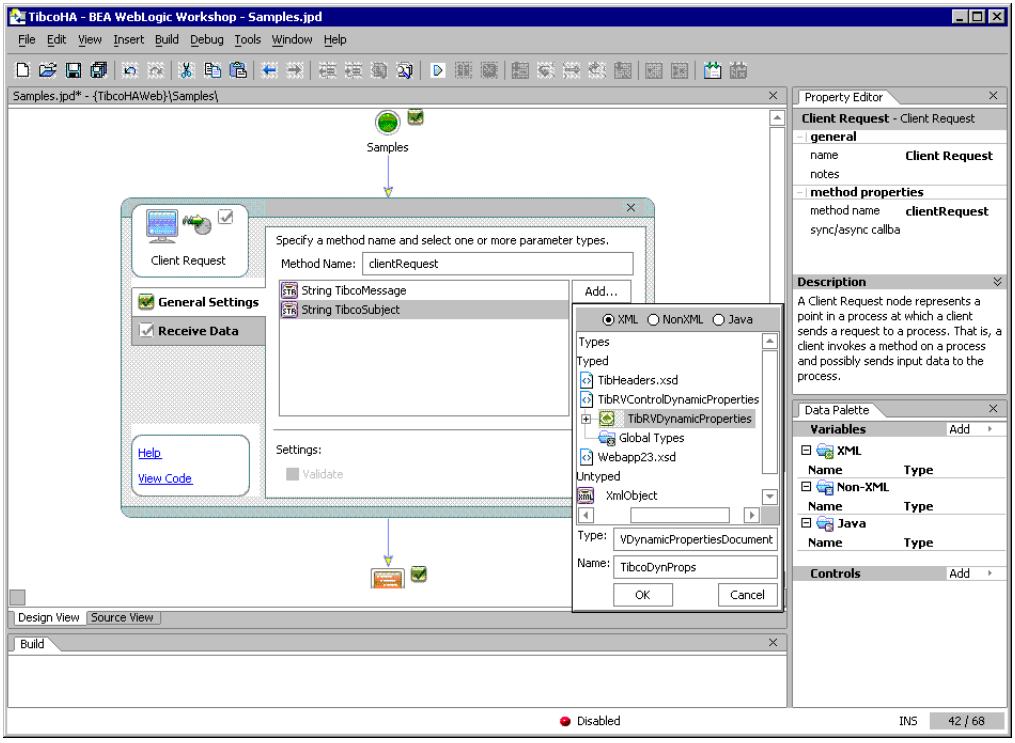
Setting Dynamic Properties

You can change the TIBCO RV control properties dynamically at runtime. The TIBCO RV control properties that you can modify are specified in the `TibRVDynamicPropertiesDocument` type document. This document conforms to the TIBCO RV Control Dynamic Properties schema, which is available in the `TibRVSchemas.jar` file.

The following is an example on how to change properties dynamically.

1. Open the **Client Request** node, as shown in the following figure.

Figure 1-5 Defining a Method Using Dynamic Properties



2. In the **General Settings** tab, add a variable of type **`TibRVDynamicPropertiesDocument`**.

3. In the **Receive Data** tab, create a new variable for the parameter that you previously created in the **General Settings** tab by entering a name for the variable. The variable type is already pre-defined based on the parameter to which you are assigning the variable.

4. Drag and drop the **setXMLProperties** function from the **Controls** tab of the **Data Palette**, into your business process.

5. Open the **Send Data** tab of the **setXMLProperties** function node. From the **Select variables to assign** drop-down list, assign the variable that you created in the **Receive Data** tab of the **Client Request** node to the corresponding parameter of the **setXMLProperties** function listed in the **Control Expects** column. All TIBCO RV Control send message operations (following the `setXMLProperties` function in the business process) using the properties you specified in the `TibRVControlDynamicPropertiesDocument`.

6. While executing your business process at runtime, provide the `TibRVControlDynamicPropertiesDocument` as input.

Schema of TIBCO RV Control Dynamic Properties

```
<?xml version="1.0"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.bea.com/wli/control/TibRVControlDynamicProperties.xsd
"
targetNamespace="http://www.bea.com/wli/control/TibRVControlDynamicPropert
ies.xsd"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xs:element name="TibRVDynamicProperties">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="service" type="xs:string" minOccurs="0"
maxOccurs="1"/>
            <xs:element name="network" type="xs:string" minOccurs="0"
maxOccurs="1"/>
            <xs:element name="daemon" type="xs:string" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="useCM" type="xs:boolean" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="cmName" type="xs:string" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="ledgerName" type="xs:string" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="requestOld" type="xs:boolean" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="syncLedger" type="xs:boolean" minOccurs="0"
maxOccurs="1"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Sample TIBCO RV Control Dynamic Properties Document

The following is a sample TIBCO RV Control document. You must provide this document at runtime when you execute your business process:

```
<tib:TibRVDynamicProperties>
   <!--Optional:-->
   <tib:service>7500</tib:service>
   <!--Optional:-->
   <tib:network>beaserv1</tib:network>
   <!--Optional:-->
   <tib:daemon>beaserv1:7500</tib:daemon>
   <!--Optional:-->
   <tib:useCM>true</tib:useCM>
   <!--Optional:-->
   <tib:cmName>cmname.runtime</tib:cmName>
   <!--Optional:-->
   <tib:ledgerName>c:/file.txt</tib:ledgerName>
   <!--Optional:-->
   <tib:requestOld>false</tib:requestOld>
   <!--Optional:-->
   <tib:syncLedger>false</tib:syncLedger>
   </tib:TibRVDynamicProperties>
```

CHAPTER 2

TIBCO Rendezvous Event Generator

TIBCO® Rendezvous™ (a product from TIBCO: www.tibco.com) enables exchange of data across applications running on distributed platforms. TIBCO Rendezvous (TIBCO RV) Event Generator is one of the WebLogic Integration™ event generators that you can create from the WebLogic Integration Administration Console. The TIBCO RV event generator listens for messages on a subject and raises events to the message broker on receiving the desired message.

Note: The TIBCO RV event generator (EG) is available in WebLogic Workshop™ only for licensed users of WebLogic Integration.

Disclaimer

Use of the TIBCO RV control and event generator with BEA WebLogic Integration in no manner confers or grants the right to use TIBCO Rendezvous including "dynamic libraries". In order to use such TIBCO products, the user of the TIBCO RV control and event generator must obtain a valid license from TIBCO. See `http://www.tibco.com` for information on how to obtain a licensed copy of Rendezvous.

Topics Included in This Section

Overview: TIBCO RV Event Generator
Describes the function of the TIBCO RV event generator within WebLogic Integration.

Creating and Using the TIBCO RV Event Generator
Describes how to create an event generator and to use it for receiving messages.

Overview: TIBCO RV Event Generator

The TIBCO RV event generator enables WebLogic Integration generate events to message broker channels. The messages are received in most formats supported by Rendezvous, converted to binary and then published to the WebLogic Integration message broker.

Using TIBCO RV event generator, you can receive messages over the base Rendezvous and certified messaging (CM) transports. Certified messages can be received in either single or distributed queues, with support for clustering in the distributed queues processing. Receipt of each message can be acknowledged by implicit or explicit confirmation.

To learn more, see Event Generators in *Managing WebLogic Integration Solutions*, which is located at the following URL:

`http://edocs.bea.com/wli/docs85/manage/evntgen.html`

Clusters

TIBCO RV event generators can be deployed on a cluster with load balancing, high availability and failover features. They use distributed queues to support these clustering features. All event generators are automatically deployed on all managed servers from the WebLogic Integration Administration Console.

TIBCO RV event generators on a cluster subscribe to a subject with a single RV daemon machine, a distributed queue option and a distinct CM name entered at the time of event generator creation. Distributed queues work with the concept of scheduler and workers. One of the queues will act as a scheduler and others as workers.A scheduler distributes the messages to workers on a round robin basis, making sure the message is received by one and only one worker. This is also referred to as the “once and only once delivery”.

Load balancing

The scheduler node sends messages to the worker nodes on a round robin basis. This is done by checking for pending tasks at individual workers end. Depending on the which worker is relatively free, the scheduler will assign the task. This is termed as load balancing.

High Availability

This implies that at any given instance, a worker and a scheduler node is always available. If a worker node goes down with a managed server, another worker will be available. If a scheduler node goes down with a managed server, a worker will take over the role of the scheduler node.

Failover

This indicates that even if a worker node goes down before acknowledging receipt of message, the scheduler node will re-assign the task to the next available worker node.

Prerequisites to Using the TIBCO RV Event Generator

Before adding the TIBCO RV Event Generator to the WebLogic Platform, perform the following:

1. Install and configure Rendezvous on your machine.
2. Deploy the TIBCO RV event generator application file **`TibRVEG.ear`** (available in `$WL_HOME/integration/lib` directory) into the integration domain template.

After successful deployment, you will be able to create event generators from the WebLogic Integration console.

Note: In a cluster environment, deploy the `TibRVEG.ear` file on the admin server only.

Creating and Using the TIBCO RV Event Generator

Perform the following steps to create a TIBCO RV event generator.

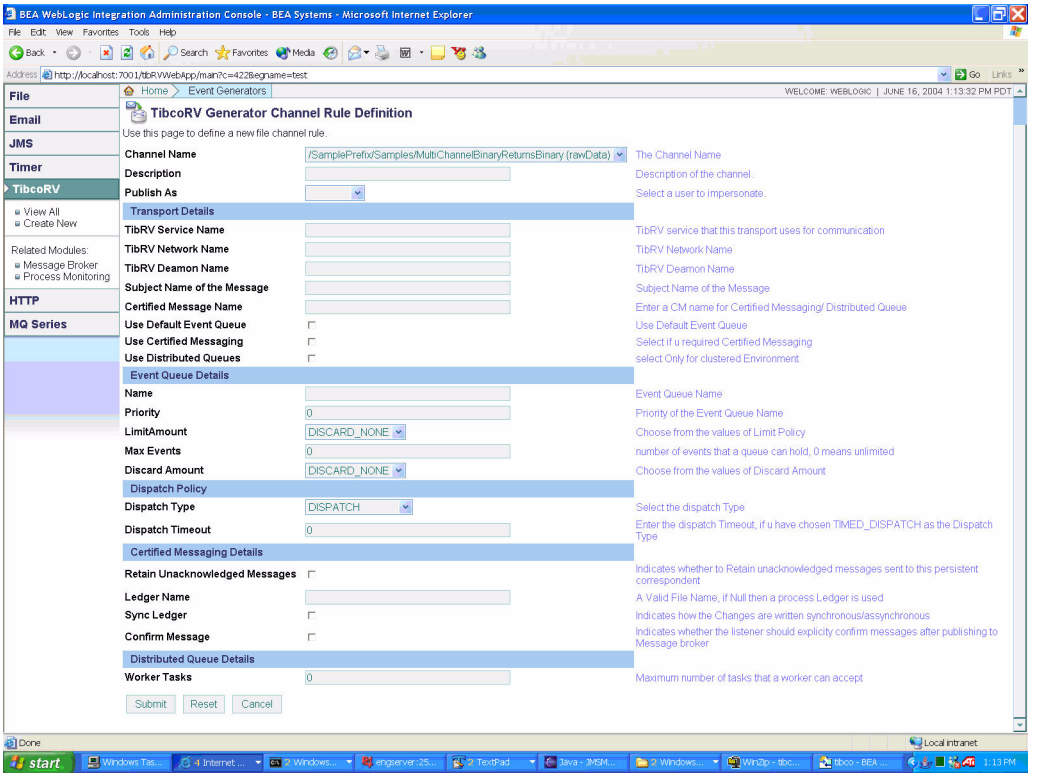
1. Enter the following URL in your html browser:

`http://localhost:`**`port`**`/wliconsole`

   Replace **`port`** with the specific port id, for example `7001`.

2. In the WebLogic Integration Administration Console home page, click **Event Generators** to display the Event Generators home page.
3. Click **TIBCO RV** option in the left frame and select **Create New** option that appears below it, to create an event generator.
4. In the **Generator Name** field, enter a unique name for the new event generator and click **Submit**.
5. In the next frame, click **Define a New Channel Rule** to display a form as shown in the following figure.

Figure 2-1 Creating a TIBCO RV Event Generator: Channel Rule Definition



6. Enter the desired information in the fields; a brief description of each is displayed adjacent to the field. Refer TIBCO Rendezvous documentation for information on Rendezvous transport, event queues and certified messaging parameters.

   Note: Select **Use Distributed Queues** option for the event generator to work on a cluster. Doing this would also make it mandatory to specify a CM name.

7. Click **Submit** to effect creation of the channel rule and the event generator.

Note: While creating an event generator, if incorrect Rendezvous related values are entered (like network, daemon and so on), the event generator is created but, a runtime exception error will be displayed.

These Rendezvous properties are not verified at the form submission stage. These values are used only when the application attempts to connect to the specified Rendezvous daemon; hence, the runtime error.

On successful creation of an event generator, a `WLI_TIBRV_`**`event-gen`**`.jar` file is created in the WebLogic server domain folder. Here, **`event-gen`** is the unique name of the event generator as specified in step 4 above. This file connects to the Rendezvous daemon, as specified in the Channel Rule Definition form, and creates a listener on the subject.

Note: Always create a single rule definition for each unique event generator. Whenever an event generator is created with multiple channel rule definitions, only the first channel rule definition is recorded and used.

Explicit Confirmation

Explicit confirmation directs the listener to explicitly confirm delivery of message after publishing to the message broker. To employ the explicit confirmation feature, select the **Confirm Message** option while defining the rules for an event generator.

If **explicit confirmation** is selected in the OAM console, the TIBCO RV event generator will confirm the message only on successful completion of the subscribed JPDs.

Notes: JPDs need to have a synchronous subscription to confirm explicitly.

If any of the subscribed JPDs throw an exception error, the TIBCO RV event generator will not confirm the message.

Retrieving Messages

Once the event generator has been created, it will start publishing messages on the subscribed subject to the WebLogic Integration message broker channel. Applications subscribed to that channel will receive the messages in raw data format. To retrieve the content of the message, insert a perform node with the following code:

```
TibrvMsg RecvdMsg = new TibrvMsg(receivedData.byteValue());
```

Where **`receivedData`** is of type `com.bea.data.RawData` and contains the message published by the TIBCO RV event generator.

In addition, you need to edit the following annotation in the JPD file:

```
/**
* @jpd:mb-static-subscription channel-name="/Soak/reply/TibcoDataChannel"
message-body="{x0}"
*/
```

to read as follows:

```
/**
* @jpd:mb-static-subscription channel-name="/Soak/reply/TibcoDataChannel"
```

```
message-body="{x0}" message-metadata="{x1}"
*/
```

Important: The parameter used in the annotation also needs to be added to the method definition. For an example, refer to Retrieving Information From a TIBCO RV Event Generator Header.

TIBCO RV Event Generator Header

This section provides the schema of a TIBCO RV event generator header and an example code to retrieve information from the header.

Schema of a TIBCO RV Event Generator Header

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.bea.com/wli/control/TibHeaders"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.bea.com/wli/control/TibHeaders"
elementFormDefault="qualified">
   <xs:element name="TibHeaders">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="ReplySubject" type="xs:string" minOccurs="0"
maxOccurs="1"/>
            <xs:element name="SendSubject" type="xs:string" minOccurs="0"
maxOccurs="1"/>
            <xs:element name="TibrvTransport" type="Transport"
minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
      <xs:complexType name="Transport">
         <xs:sequence>
         <xs:element name="Service" type="xs:string" minOccurs="0"
maxOccurs="1"/>
         <xs:element name="Network" type="xs:string" minOccurs="0"
maxOccurs="1"/>
         <xs:element name="Daemon" type="xs:string" minOccurs="0"
maxOccurs="1"/>
         </xs:sequence>
```

```
      </xs:complexType>
</xs:schema>
```

Retrieving Information From a TIBCO RV Event Generator Header

Following is a code example for retrieving **`replySubject`** from the TIBCO RV event generator header.

```
/**
* @jpd:mb-static-subscription channel-name="/Soak/reply/TibcoDataChannel"
message-body="{x0}" message-metadata="{x1}"
*/
public void subscription(com.bea.data.RawData x0, com.bea.xml.XmlObject x1)
{
   //#START: CODE GENERATED - PROTECTED SECTION - you can safely add code
above this comment in this method. #//
   // input transform
   // parameter assignment
   this.recievedBytes = x0;
   this.Header = x1;
   //#END  : CODE GENERATED - PROTECTED SECTION - you can safely add code
below this comment in this method. #//
   TibHeaders
tibHeader=TibHeadersDocument.Factory.newInstance().addNewTibHeaders();
      tibHeader.set(this.Header);
   String replySubject=tibHeader.getReplySubject();
}
```

CHAPTER 3

TIBCO Rendezvous Sample

This chapter gives you a complete understanding of TIBCO® RV control and event generator with the help of a real-life application. The sample application provided is working code and implements a simple loan processing application that uses request/reply, certified messaging and TIBCO event generator headers.

Note: This sample application is provided for your convenience and is not supported by BEA.

What the Sample Does

The application works in the following manner:

1. The **`LoanRequest`** process will send a loan request message through a TIBCO RV control with subject `loan.request`.
2. The **`Bank`** process will listen to messages (of subject `loan.request`) through a TIBCO RV event generator and reply to these messages as approved.
3. In processing the loan request, the **`Bank`** process will send certified messages to `loan.process` and the **`ProcessDesk`** process will receive these messages and process the loan request.

Note: It is important that you read through Chapter 1, "TIBCO Rendezvous Control," and Chapter 2, "TIBCO Rendezvous Event Generator," before you proceed.

Prerequisites to Using the Sample Application

Before you start using the sample application, you need to ensure that you can create TIBCO RV controls and event generators. Use the following as a checklist.

1. Ensure the TIBCO RV event generator application (`TibRVEG.ear`) is deployed in the integration domain template, and TIBCO EG is displayed in the event generators tab of the WebLogic Integration™ Administration Console.
2. Ensure the `tibrvj.jar` file from the TIBCO® Rendezvous™ installation is imported into the Libraries folder of the application where the TIBCO RV control is used.
3. Ensure the `TibRVControl.jar` and `TibRVSchemas.jar` files are copied to the Libraries folder of the application.
4. Ideally, build this sample application in **development** mode. However, if you are in **production** mode, turn on the `testConsoleFlag`.

Getting Started With the Sample Application

The sample application `RvSample.zip` is available at the following URL:

https://wli8.projects.dev2dev.bea.com/servlets/ProjectDocumentList?folderID=57&expandFolder=57&folderID=0

The sample file is listed as **TIBCO Rendezvous Control and Event Generator Sample**. Download the `RvSample.zip` file and perform the following steps to integrate and use the application.

1. Unzip the `RvSample.zip` file to extract all the application code files. These files are automatically extracted under a `RvSample` directory.
2. Open the **`RvSample.work`** file in WebLogic Workshop™.
3. Create two TIBCO RV event generators in the WebLogic Integration Administration Console. You can provide any names for these two event generators, as long as they are unique. Specify the following values for the respective event generators.

   `Subject`: `loan.request`
   `Channel`: `/loan/request/EG`

   `Subject`; `loan.process`
   `Channel`: `/loan/request/backendProcess`
   `CM`: `true`

Leave the rest of the parameter fields blank.

4. Press **F7** to build the application in WebLogic Workshop, which should also automatically deploy it.
5. On successful build, execute the **`LoanRequest`** process by pressing **Ctrl+F10**.
6. After successful execution of the application, the WebLogic console displays the following statements, one for each of the three processes:

```
Loan Request: Loan Approved
Bank: Loan is successfully getting processed at the process desk
Process Desk: Received the loan request... will process
```

Index