SYBASE®
An SAP® Company

Configuration and Users Guide

RepConnector™ 15.0.2

DOCUMENT ID: DC01988-01-1502-01
LAST REVISED: August 2013

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

RepConnector

RepConnector™ delivers database events and metadata from Replication Server® to the configured destination. It provides a nonpolling, nontrigger-based solution to database integration, building on the Replication Server noninvasive system to push database transactions into a traditional integration environment.

With RepConnector, you need not poll a database for changes, or to add triggers for event notification. Configuring RepConnector requires the configuration of other servers and software in your RepConnector environment.

RepConnector provides adapters for sending messages to TIBCO, IBM MQ, SonicMQ, or Sybase® certified Java Message Service (JMS) 1.1 compliant messaging systems. It can also send messages to virtually any user or application.

RepConnector:

- Follows transactional behavior
- Manages connections using:
  - RepConnector Manager – a GUI in the Eclipse framework
  - **ratool** – a command line utility
- Can group database events into a single transaction
- Supports `text` and `image` datatypes
- Parses replication events and generates XML documents
- Transforms incoming database events into XML messages, and routes them into configured message queues
- Can transform incoming database events into an appropriate application-specific format
- Can route incoming database events to any destination
- Detects message events and routes them to database tables
- Supports these application servers:
  - JBoss AS 6.1.0.Final “Neo”
  - JBoss AS 7.1.1.Final “Brontes”
  - Oracle WebLogic Server 12c (12.1.1)

RepConnector Architecture

RepConnector is based on the JCA (Java Connector Architecture) 1.5 specification of Java EE 6. It runs in a Java EE-compliant application server environment.

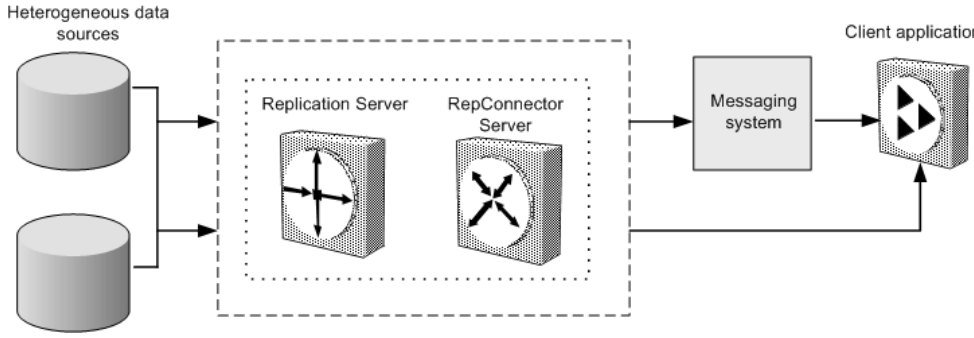
The architecture consists of three modules:

- Event Capture – listens for events from Replication Server or from the messaging system. Event Capture provides a TCP socket that listens for Replication Server events, and acts as a client for the messaging system, listening on the message bus for messaging events.
- Event Transformation – transforms an event before it is routed to its destination. In real-time messaging, RepConnector transforms the event to XML. You can also customize the module to add a customized transformer plug-in. When you send a message to the database, RepConnector transforms the event to a SQL statement.
- Event Sender – by default, routes the event to a messaging system or to a database. You can customize the Event Sender to send events to virtually any endpoint.

For real-time messaging, RepConnector uses Replication Server technology to detect business events as they occur in the database. Upon receiving events from Replication Server, RepConnector transforms those events to XML-formatted messages, then sends the XML messages to the configured messaging systems. RepConnector guarantees that the message routing is transactional.

In the reverse direction, RepConnector detects events from any of the supported messaging systems, transforms those events to SQL statements, and sends them to the configured database. These incoming events are either SQL commands or an XML representation of SQL commands.

Figure 1: RepConnector Architecture



RepConnector Process Flow

You can use RepConnector to route database events from Replication Server to messaging systems, or to route events from messaging systems to database tables.

Routing Database Events from Replication Server to Messaging Systems

1. When an event occurs in a database, Replication Server detects the event and pushes it to Event Capture module, which is listening for such events.
2. When the Replication Server event arrives from the Event Capture module, the Event Transformation module transforms the event into XML.

To transform messages into an application-specific format, develop your own transformation module to replace the default XML transformation.

3. After the message is transformed to XML, the Message Sender Module sends the XML message to the configured message system.

   You can develop your own sender class to route the message to other destinations.

Routing Events from Messaging Systems to Database Tables

1. The Event Capture module listens for messages arriving in the configured messaging system.
2. The Event Capture module receives messages and triggers the Event Transformation module.
3. The Event Transformation module analyzes messages and, if necessary, transforms them to SQL format.
4. The Message Sender module applies any transformed SQL statements to the database.

See also

- *Customizing the Sender and Formatter Processors* on page 65

Guaranteed Delivery

When routing replication events to messaging systems, RepConnector works with Replication Server to guarantee delivery.

RepConnector receives a message from Replication Server, and writes the message transaction ID to a file, ensuring that no transaction is lost in the event of a software failure. RepConnector then attempts to deliver the message. RepConnector sends an acknowledgment when the message is successfully delivered to the specified messaging system. This process guarantees that no transactions are skipped, and that deliveries are sequential.

Status and Error Reporting

When routing messaging events to database tables, RepConnector reports status and errors through a status queue. Client applications can monitor the status queue and retrieve any status or error messages that occur throughout the process.

Configuring RepConnector (Overview)

Configure each RepConnector component before creating RepConnector connections. RepConnector components include Sybase products as well as third-party products.

1. Set up Replication Server to send replicated events to RepConnector. See *Configuring Replication Server for RepConnector* on page 5.

2. Configure your database server and messaging system in the RepConnector environment. See *Configuring RepConnector* on page 19.
3. Create and configure your RepConnector connection. See *Get Started with RepConnector Manager* on page 15 and *Configuring RepConnector* on page 19.
4. Manage runtime control. See *Managing a RepConnector Connection* on page 37 for information about how your configuration of the RepConnector environment can influence your runtime control and management of RepConnector connections.

Configuring Replication Server for RepConnector

Establish the RepConnector connection in Replication Server before configuring RepConnector.

Prerequisites

1. Configure a Replication Server environment.
2. Add the primary database to the replication system (including updating the interfaces file that contains the connection information for the database server).
3. Mark the primary tables and procedures for replication.

If you have not completed these tasks, see the *Replication Server Configuration Guide* before you proceed.

Task

1. In the Replication Server interfaces file, add an entry for RepConnector.
2. Verify that Replication Server is running.
3. In Replication Server:
   - Create a database connection to communicate with RepConnector.
   - Create a replication definition to identify the data to be replicated.
   - Create a subscription to identify the location to which the data will be replicated.
4. Resume the database connection.

As you work through the Replication Server configuration sections, use the configuration worksheet to record the values used to configure the RepConnector connection to Replication Server.

See also

- *Configuration Worksheets* on page 93

Updating the Interfaces File

Add a RepConnector connection entry to the Replication Server interfaces file for each RepConnector connection. The interfaces file contains network information that Replication Server requires to connect to RepConnector.

Either record this information on the worksheet provided as you go through the procedure, or complete the worksheet first, then use it in the procedure.

- Server name – the name of the data server. This name should be unique and case-sensitive. Record this value on line 3.a of the worksheet.

  This is also the RepConnector connection DSI name, which you will need later when you configure the RepConnector connection.

  Note: Sybase recommends that you use a name that clearly identifies this connection as allowing Replication Server to communicate with RepConnector, and that distinguishes it from a traditional connection between any data server and the corresponding database.
- Protocol – the network protocol for the DSI connection. Record this value on line 3.b of the worksheet.

  You can use either the Transmission Control Protocol (TCP) or the NLWNSCK protocol on Windows, and either TCP or the Transport Layer Interface (TLI) TCP protocol on UNIX.
- Host name – the machine name where the RepConnector connection will be running. This value should be recorded on line 3.c of the worksheet.
- Port Number – the port where the RepConnector connection will be listening. This must be an unused port number on the host machine. Record the value on line 3.d of the worksheet.

Adding a RepConnector Entry to the Interfaces File

Create a new entry for the RepConnector connection in the interfaces file, on the machine on which Replication Server is running.

1. Use **dsedit**, a utility that is part of the Replication Server installation, located in the `OCS-15_0\bin` subdirectory on Windows, and `OCS-15_0/bin` on UNIX.

   Note: You can manually add information to the interfaces file, but Sybase recommends that you use **dsedit**.

   See the *Adaptive Server Utility Guide* for more information about the **dsedit** utility and editing interfaces files.
2. Open the file to verify that your entry is correct. The Replication Server interfaces file is located in `%SYBASE%\ini\sql.ini` on Windows and `$SYBASE/interfaces` on UNIX, where `%SYBASE%` and `$SYBASE` are the locations of the Replication Server installation.

   A sample entry for the interfaces file:
   - On Windows:

     ```
     [server_name]
     master=protocol, machine_name, port_number
     query=protocol, machine_name, port_number
     ```
   - On UNIX:

```
server_name
master protocol machine_name port_number
query protocol machine_name port_number
```

where:

- *server_name* is the DSI name as recorded on your worksheet in 3.a.
- *protocol* is the network protocol for the DSI connection as recorded on your worksheet in 3.b.
- *port_number* is the port recorded on your worksheet in 3.d.

Examples

This is an interface entry for the RepConnector connection:

- On Windows:

```
[RepConnector]
master=TCP, localhost, 7000
query=TCP, localhost, 7000
```

- On UNIX:

```
RepConnector
master tcp ether localhost 7000
query tcp ether localhost 7000
```

Note: If you are creating more than one RepConnector connection, each entry to the interfaces file must have a unique name and port number.

Creating the Connection to RepConnector in Replication Server

Create a new RepConnector connection in the Replication Server, which defines the information that Replication Server uses to connect to the RepConnector server, to replicate data.

Prerequisites

Gather this information:

- The location of the **isql** utility, which is in the Replication Server installation directory in `OCS-15\bin` on Windows or `OCS-15/bin` on UNIX.
- The DSI name for the RepConnector connection (from line 3.a on your worksheet; it is also the same name added to the interfaces file).
- A user name to connect to the RepConnector connection (from line 3.e on your worksheet).
- A password for this user name (from line 3.f on your worksheet).

Task

1. Using **isql**, log in to the Replication Server:

   ```
   isql -U <username> -P <pwd> -S <server_name>
   ```

   where:
   - *username* is the user ID with **sa** permission in the Replication Server.
   - *pwd* is the password for the user ID.
   - *server_name* is the name of the Replication Server.
2. Create the connection and define the user ID and password for the RepConnector connection:

   ```
               create connection to <dataserver>.<database>
               set error class to rs_sqlserver_error_class
               set function string class to
   rs_sqlserver_function_class
               set username <dsi_username>
               set password <dsi_password>
               set batch to 'off'
               set dsi_xact_group_size to '-1'
               set dynamic_sql to 'off'
               set dsi_bulk_copy to 'off'
               set dsi_row_count_validation to 'off'
               with dsi_suspended
   ```

   where:
   - *dataserver* is the RepConnector connection DSI name. This is the same name you added to the Replication Server interfaces file. Use the name recorded in 3.a on your worksheet.
   - *database* is the name of the database to which you are replicating. Record this value in 3.j on the worksheet.

     Note: When you create this connection in Replication Server, you must designate it with the *dataserver.database* data pair value. However, the database name in Replication Server is only a placeholder, and RepConnector does not use that name. Therefore, use a name that clearly designates the replicate or destination (in this case RepConnector), to manage the RepConnector connection in an environment where you are also managing traditional Replication Server connections, that have destinations that are actually databases. For example, designate the connection as `RepConnector.RepCondb`.
   - *dsi_username* is the user ID that is used to connect to the RepConnector connection. Use the value recorded in 3.e on your worksheet.
   - *dsi_password* is the password for the user ID. Use the value recorded in 3.f on your worksheet.
   - **set batch to 'off'** is required by RepConnector. This instructs Replication Server not to batch the commands to send to RepConnector.

- **set dsi_xact_group_size to '-1'** is required by RepConnector. This instructs Replication Server not to group the transactions as a single transaction before sending them to RepConnector.
- **set dynamic_sql to 'off'** is required by RepConnector. This instructs Replication Server not to use dynamic SQL.
- **set dsi_bulk_copy to 'off'** is required by RepConnector. This instructs Replication Server not to use the bulk-copy-in feature.
- **set dsi_row_count_validation to 'off'** is required by RepConnector. This instructs Replication Server to disable row-count validation.

Note: RepConnector does not support the batching of commands in Replication Server. If you have already created the connection, and have not set the **batch** and **dsi_xact_group_size** parameters as indicated above, use the **configure connection** and **alter connection** commands to set them. See the *Replication Server Reference Manual*.

For example:

```
create connection to RepConnector.RepCondb
set error class to rs_sqlserver_error_class
set function string class to
rs_sqlserver_function_class
set username sa
set password null
set batch to 'off'
set dsi_xact_group_size to '-1'
set dynamic_sql to 'off'
set dsi_bulk_copy to 'off'
set dsi_row_count_validation to 'off'
with dsi_suspended
```

Creating the Replication Definition in Replication Server

Create the table replication definition in the Replication Server. A replication definition describes the data that can be replicated for a table or stored procedure defined in the primary database. RepConnector supports replication of DML commands and stored procedures.

Prerequisites

Mark primary tables or stored procedures for replication. You must know the Adaptive Server® name and database name in which the primary table resides.

Task

1. Record this information on your worksheet:
   - Primary data server name (in 3.k)
   - Primary database name (in 3.j)
   - Table names and column field names

2. Create the table replication definition:

```
create replication definition
     <replication_definition_name>
with primary at
     <dataserver>.<database>
with all tables named '<table_name>'
( <column_name> <column_datatype>,
     …)
primary key (<column name>,..))
searchable columns (<column_name>,..)
```

where:

- *replication_definition_name* is the name for the replication definition.
- *dataserver* is the name of the server containing the primary data (3.k).
- *database* is the name of the database containing the primary data (3.j).
- *table_name* is the name of the primary table containing the data.
- *column_name* is the column name from the primary table.
- *column_datatype* is the datatype for the column name.
- *primary key* is a primary key, or a set of primary keys, defined in the table.

For example:

```
create replication definition authors_rep
with primary at primary_ase.pubs2
with all tables name 'authors' (
     au_id varchar(11),
     au_lname varchar(40),
     au_fname varchar(20),
     phone char(12),
     address varchar(40),
     city varchar(20),
     state char(2),
     country varchar(12),
     postalcode char(10))
primary key (au_id)
searchable columns(au_id)
```

For more information about **create replication definition**, see the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

Creating a Function Replication Definition

Create a function replication definition.

1. Record this information on your worksheet:
   - Primary data server name (line 3.k)
   - Primary database name (line 3.j)
   - Procedure and parameter names

2. Create the function replication definition:

```
create function replication definition
<replication_definition_name>
with primary at <dataserver>.<database>
deliver as '<procedure_name>' (
     <@param_name> <datatype>,
     …)
searchable parameters (<@param_name>,..>)
```

where:

- *replication_definition_name* is the name of the function replication definition.
- *dataserver* is the name of the server containing the primary data.
- *database* is the name of the database containing the primary data.
- *procedure_name* is the name of the stored procedure in the primary dataserver.
- *param_name* is the parameter name from the function.

For example:

```
create function replication definition ins_authors with primary at
primary_ase.pubs2(
     @au_id varchar(11),
     @au_lname varchar(40),
     @au_fname varchar(20),
     @phone char(12),
     @address varchar(40),
     @city varchar(20),
     @state char(2),
     @country varchar(12),
     @postalcode char(10))
)
searchable parameters(@au_id)
```

For more information about **create function replication definition**, see the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

Creating and Verifying the Subscription in Replication Server

Create a subscription and verify that it is valid both at the primary table and at the RepConnector connection in the Replication Server.

A subscription instructs Replication Server to copy data from the primary table to the specified RepConnector connection. The subscription describes the replicated information that RepConnector can accept.

1. Create the subscription using the RepConnector connection name as the parameter value for the **with replicate at** command.
2. Record this information on your worksheet:

- Name of the RepConnector connection (3.a)
- Name of the replication definition

3. Create the database subscription:

```
create subscription <subscription_name>
for <replication_definition_name>
with replicate at <dataserver>.<database>
without materialization
```

where:

- *subscription_name* is the name of your subscription.
- *replication_definition_name* is the name of the replication definition.
- *dataserver* is the name of the database connection you created to connect to RepConnector.
- *database* is the name of the database to which you are replicating.

  RepConnector does not actually use the database name; It is only a placeholder, used to meet Replication Server syntax requirements.

  Because a RepConnector connection is the destination instead of an actual database, Sybase recommends that you use a unique name that represents the RepConnector connection.

For example:

```
create subscription authors_sub
for authors_rep
with replicate at RepConnector.RepCondb
without materialization
```

4. Verify that the subscription is valid both at the primary database and the replicate database:

```
check subscription <subscription_name>
for <replication_definition_name>
with replicate at <dataserver>.<database>
```

where:

- *subscription_name* is the name of the subscription.
- *replication_definition_name* is the name of the table or function replication definition for which the subscription is created.
- *dataserver* is the name of the RepConnector connection (line 3a).
- *database* is the name of the database to which you are replicating.

  RepConnector does not actually use the database name; It is only a placeholder, used to meet Replication Server syntax requirements.

For example:

```
check subscription authors_sub
for authors_rep
with replicate at RepConnector.RepCondb
```

Resuming the Connection to RepConnector

To ensure that Replication Server replicates commands to RepConnector, resume the connection from Replication Server to RepConnector.
Enter:

```
resume connection to <dataserver>.<database>
```

where:

- *dataserver* is the name of the connection.
- *database* is the name of the database to which you are replicating. This is the same value you used when you created the connection to RepConnector in the Replication Server.

For example:

```
resume connection to RepConnector.RepCondb
```

A connection has now been established between RepConnector and Replication Server.

Note: The connection does not show an active status until the RepConnector connection has successfully started.

Get Started with RepConnector Manager

RepConnector Manager is a plug-in to the Eclipse framework that you can use to manage the activities of the RepConnector runtime components.

You can run RepConnector Manager on the local machine where the runtime component is installed, or on any remote machine that can access the machine on which the runtime component is installed. To access a remote machine, verify that the HTTP connection to the application server on which the RepConnector runtime component is deployed and running, and is available throughout the network.

See the Eclipse documentation for more information about how to use the Eclipse framework.

Starting RepConnector Manager

Run the `RepConnectorManager.bat` file on Windows or `RepConnectorManager.sh` file on UNIX to start the RepConnector Manager.

1. From a command prompt, navigate to the RepConnector Manager installation directory.
2. To start RepConnector Manager, enter:
   - On Windows:
     ```
     RepConnectorManager.bat
     ```
   - On UNIX:
     ```
     RepConnectorManager.sh
     ```
3. Select a workspace and click **OK**.

   To define the specified workspace as the default workspace, select **Use this as the default and do not ask again**.

   The Sybase RepConnector Manager Welcome window appears; if it does not, select **Help > Welcome**.

Displaying the RepConnector Manager View

Display the RepConnector Manager view.

1. To display the RepConnector Manager view, perform one of these actions:
   - On the Welcome window, click the RepConnector Manager icon, or,
   - From the Eclipse menu bar:
     a. Select **Windows > Show View > Other**.

   b. In the Show View dialog box, select **Sybase** > **RepConnector Manager**.
2. Resize or relocate the RepConnector tree view within the Eclipse workbench, if needed.

Managing Connection Profiles

Use connection profiles to manage the RepConnector connections defined for an installed RepConnector instance running on an application server.

A RepConnector profile contains the connection property information needed to connect to a RepConnector runtime instance. You can configure as many RepConnector profiles as necessary to manage all the RepConnector installations from a single RepConnector Manager.

The RepConnector Manager installation includes two sample default profiles you can use to connect to your RepConnector runtime:

- JBoss:8080 for JBoss Application Server
- weblogic:7001 for WebLogic Server

Creating a New Profile

Create a new RepConnector profile.

1. From the RepConnector Manager view, right-click the *Sybase RepConnector* folder, and select **New Connection Profile**.
2. Enter profile details:
   - Profile Name – the name must be unique within this instance of RepConnector Manager. The name of the profile can contain alphanumeric characters, colons, periods, hyphens, and underscores. It cannot contain white spaces.
   - Host – the HTTP host name of the machine where the target RepConnector runtime is running. The default value is localhost.
   - Port – the HTTP port number where the target RepConnector runtime is listening. The default is 8080 for JBoss and 7001 for WebLogic.
   - User – the RepConnector administrator user ID to connect to the RepConnector server. The default is repraadmin.
3. Select **OK** to create a new profile.

   For example, enter:

```
Profile Name: JBoss:8080
Host: myhost
Port: 8080
User: repraadmin
```

See also

- *Setting Up the RepConnector Server Administrator Login* on page 17

Renaming a Profile

Rename a RepConnector profile.

1. Right-click the profile to rename, and select **Rename Profile**.
2. Enter a new profile name that is unique within the same RepConnector Manager instance.
3. Select **OK**.

Editing the Profile Properties

Modify RepConnector profile properties.

1. Right-click the profile to modify, and select **Edit Profile Properties**.
2. Modify the fields and select **OK**.

Deleting a Profile

Delete a RepConnector profile.

1. Right-click the profile and select **Delete Profile**.
2. Select **OK**.

Note: This deletes a profile only from the RepConnector Manager tree view only; it does not make changes to the runtime configuration.

Setting Up the RepConnector Server Administrator Login

Configure the administrator login for a RepConnector Server.

Each RepConnector instance has one administrator login, which is, by default, “repraadmin” with no password. To secure access to the RepConnector runtime, Sybase recommends that you change this login and password.

- The login name and the password must be alphanumeric.
- The length of the password must be must be equal to or less than 30 characters.

1. From the command prompt, navigate to:

   On Windows:

   ```
   <AppServer_install_dir>\repra\bin
   ```

   On UNIX:

   ```
   <AppServer_install_dir>/repra/bin
   ```

2. To change the user name and password:

   On Windows, enter:

   ```
   setlogin.bat <old_user_name> <old_password> \
   <new_user_name> <new_user_password>
   ```

On UNIX, enter:

```
setlogin.sh <old_user_name> <old_password> \
<new_user_name> <new_user_password>
```

Note: Having no password is represented by empty quotation marks. To create a password, enter:

```
setlogin.sh repraadmin ""\ repraadmin thepassword
```

Logging In to the RepConnector Server

Log in to the RepConnector Server.

1. Start the RepConnector Server.
2. Right-click the connection profile and select **Login**.
   You see a window that lists all of the RepConnector profile properties and an empty password field.
3. Enter the password and click **Login**.

Note: For security reasons, you must reenter the password each time you log in.

Once you have successfully logged in, you can see all the connections that are configured with the RepConnector runtime component. RepConnector provides one default connection named “sample_repConnector.”

Refreshing the Profile

Refresh the tree view under the profile folder to show the current status of the runtime. Right-click the profile and select **Refresh View**.

Logging Out of the Profile and the RepConnector Runtime

Log out of the RepConnector profile to disconnect a profile.

1. Right-click the logged-in profile and select **Logout**.
2. Select **OK**.

Configuring RepConnector

Configure your environment before you configure the RepConnector connection.

Configuring RepConnector Connections

Configure a RepConnector connection to listen for events from a database and route the events to a messaging system, or to listen for events from a messaging system and route the events to a database.

Prerequisites

- Start the application server and set up the messaging system.
- Start RepConnector Manager, create a connection profile, and connect to a RepConnector runtime instance.

Task

Configure a RepConnector connection to listen from events from:

- A database, then route the events to a messaging sytem – select **REPLICATION** as the inbound type and one of the messaging systems (JMS, TIBCO, IBMMQ) as the outbound type.
  The inbound type is the source from which the RepConnector Connection receives data. The outbound type is the destination to which the RepConnector connection routes the data.
- A database, then route the events to a user-defined sender processor (a file, for example) – select **REPLICATION** as the inbound type and **CUSTOM** as the outbound type.
- A messaging system, then route the events to a database – select one of the messaging systems (JMS, TIBCO, IBMMQ) as the inbound type and **DATABASE** as the outbound type.

See also

- *Get Started with RepConnector Manager* on page 15

Configuring the RepConnector for Messaging Systems

Before validating RepConnector connection, you must configure your RepConnector environment, and restart your application server. This enables you to ping the messaging system to verify that it is configured correctly.

If you do not configure your RepConnector environment before configuring the RepConnector connection, RepConnector Manager allows you to create the connection even

if the ping fails. However, you must still verify the connection once you have created the RepConnector environment.

Note: You can skip this task if you are using JBoss HornetQ JMS or WebLogic JMS.

To set the environment for the messaging system, modify the `%REPRA_HOME%\bin\repra_env.bat` file on Windows, and `$REPRA_HOME/bin/repra_env.sh` file on UNIX , where `%REPRA_HOME%` or `$REPRA_HOME` is the RepConnector installation directory.

Configuring RepConnector for SonicMQ JMS Messaging Systems

Set up the RepConnector environment to enable RepConnector to communicate with SonicMQ JMS.

Prerequisites

Verify that the path to the SonicMQ library files are defined correctly in the RepConnector environment batch or script files.

Task

1. Verify that the line in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that defines the SONICMQ_HOME environment variable is not commented out, and that it points to the installation location for SonicMQ. For example:
   - On Windows:
     ```
     set SONICMQ_HOME=C:\SonicSoftware\SonicMQ
     ```
   - On UNIX:
     ```
     SONICMQ_HOME= /work/SonicSoftware/SonicMQ
     ```
2. Verify that the lines that define the directory structure for the SonicMQ library file, `sonic_Client.jar`, are not commented out and that they are correct for your environment.

   On Windows:

   ```
   CLASSPATH=%CLASSPATH%;%SONICMQ_HOME%\lib\sonic_Client.jar
   BOOTCLASSPATH=%BOOTCLASSPATH%;%SONICMQ_HOME%\lib\sonic_Client.jar
   ```

   On UNIX:

   ```
   REPRA_CLASSPATH=$SONICMQ_HOME/lib/sonic_Client.jar
   CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
   BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
   ```
3. Restart your application server.

Configuring RepConnector for TIBCO

Set up the RepConnector environment so that RepConnector can communicate with TIBCO RV, TIBCO RVCM, TIBCO AECM, and TIBCO Enterprise for JMS.

Configuring TIBCO RV, RVCM

Set up the RepConnector environment to communicate with TIBCO RV and RVCM.

1. Verify that the line in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that defines the TIBCO_HOME environment variable is not commented out, and that it points to the installation location for TIBCO Rendezvous. For example:

   On Windows:

   ```
   TIBCO_HOME=c:\tibco71
   ```

   On UNIX:

   ```
   TIBCO_HOME=/work/tibco71
   ```

2. Verify that the lines that define the directory structure for the TIBCO Rendezvous library file, `tibrvj.jar`, are not commented out, and that they are correct for your environment. For example:

   On Windows:

   ```
   REPRA_CLASSPATH=%TIBCO_HOME%\lib\tibrvj.jar
   set CLASSPATH=%CLASSPATH%;%REPRA_CLASSPATH%
   BOOTCLASSPATH=%BOOTCLASSPATH%;%REPRA_CLASSPATH%
   ```

   On UNIX:

   ```
   REPRA_CLASSPATH=$TIBCO_HOME/lib/tibrvj.jar:$REPRA_CLASSPATH
   CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
   BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
   ```

3. Restart your application server.

Configuring TIBCO AECM

Set up the RepConnector environment to communicate with TIBCO AECM.

1. Verify that the lines in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that define the TIBCO_HOME environment variable are not commented out, and that they point to the installation location for TIBCO Active Enterprise. For example:

   On Windows:

   ```
   TIBCO_HOME=c:\tibco71
   ```

   On UNIX:

   ```
   TIBCO_HOME=/work/tibco71
   ```

2. Verify that the lines that define the directory structure for the TIBCO Active Enterprise Certified Messaging library files, `Maverik4.jar` and `TIBRepoClient4.jar`, are not commented out and that they are correct for your environment.

   On Windows:

   ```
   REPRA_CLASSPATH=%TIBCO_HOME%\Adapter\SDK\java\Maverick4.
   jar:
   REPRA_CLASSPATH=%REPRA_CLASSPATH%:%TIBCO_HOME%\Adapter\
   SDK\java\TIBRepoClient4.jar
   CLASSPATH=%CLASSPATH%:%REPRA_CLASSPATH%;
   BOOTCLASSPATH=%BOOTCLASSPATH%:%REPRA_CLASSPATH%
   ```

   On UNIX:

   ```
   REPRA_CLASSPATH=$TIBCO_HOME/Adapter/SDK/java/Maverick4.
   jar:$REPRA_CLASSPATH
   REPRA_CLASSPATH=$REPRA_CLASSPATH:$TIBCO_HOME/Adapter/SDK/
   java/TIBRepoClient4.jar
   CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
   BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
   ```

3. Restart your application server.

Configuring TIBCO Enterprise for JMS

Set up the RepConnector environment to communicate with TIBCO Enterprise for JMS.

1. Verify that the lines in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that define the TIBCO_HOME environment variable are not commented out, and that they point to the installation location for the TIBCO Enterprise for JMS environment. For example:

   On Windows:

   ```
   TIBCO_HOME=c:\tibco71
   ```

   On UNIX:

   ```
   TIBCO_HOME=/work/tibco71
   ```

2. Verify that the lines that define the directory structure for the TIBCO Enterprise for JMS library files (`tibrvjms.jar`, `tibjms.jar`, `jms.jar`) are not commented out and that they are correctly defined for your environment.

   On Windows:

   ```
   REPRA_CLASSPATH=%REPRA_CLASSPATH%;%TIBCO_HOME%\JMS\
   Clients\java\jms.jar
   REPRA_CLASSPATH=%REPRA_CLASSPATH%;%TIBCO_HOME%\JMS\
   Clients\java\tibrvjms.jar:
   $REPRA_CLASSPATH
   REPRA_CLASSPATH=%REPRA_CLASSPATH%;%TIBCO_HOME%\JMS\
   Clients\java\tibjms.jar:
   CLASSPATH=%CLASSPATH%:%REPRA_CLASSPATH%
   BOOTCLASSPATH=%BOOTCLASSPATH%:%REPRA_CLASSPATH%
   ```

   On UNIX:

```
REPRA_CLASSPATH=$TIBCO_HOME/jms/clients/java/jms.jar
REPRA_CLASSPATH=$TIBCO_HOME/jms/clients/java/tibrvjms.jar:
$REPRA_CLASSPATH
REPRA_CLASSPATH=$TIBCO_HOME/jms/clients/java/tibjms.jar:
$REPRA_CLASSPATH
CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
```

3. Restart your application server.

Configuring RepConnector for IBM WebSphere MQ

Set up the RepConnector environment so that RepConnector can communicate with IBM WebSphere MQ.

1. Verify that the lines in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that define the IBMMQ_HOME environment variable are not commented out and that they point to the installation location for the IBM WebSphere MQ environment. For example:

   On Windows:

   ```
   IBMMQ_HOME=c:\Program Files\IBM\WebSphere MQ
   ```

   On UNIX:

   ```
   IBMMQ_HOME=/opt/mqm
   ```

2. Verify that the lines that define the directory structure for the IBM WebSphere MQ library files, `mq.jar` and `mqbind.jar`, are not commented out and that they are defined correctly for your environment.

   On Windows:

   ```
   REPRA_CLASSPATH=%IBMMQ_HOME%\java\lib\com.ibm.mq.jar;
   REPRA_CLASSPATH=%IBMMQ_HOME%\java\lib\com.ibm.mqbind.jar:
   %REPRA_CLASSPATH%
   CLASSPATH=%CLASSPATH%:%REPRA_CLASSPATH%
   BOOTCLASSPATH=%BOOTCLASSPATH%:%REPRA_CLASSPATH%
   ```

   On UNIX:

   ```
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib/com.ibm.mq.jar
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib/com.ibm.mqbind.jar:
   $REPRA_CLASSPATH
   CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
   BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
   ```

3. If you are connecting to the remote MQ daemon, verify that the MQ Server environment variable is defined correctly in the `repra_env.bat` file on Windows, or the `repra_env.sh` file on UNIX. If the environment variable is not defined correctly, modify it as necessary.

   On Windows:

   At the command line, where CHANNEL1 is the name you have defined for the channel for the server connection, enter:

```
set MQSERVER=CHANNEL1/TCP/'mymachine(1414)'
```

On UNIX:

At the command line, where CHANNEL1 is the name you have defined for the channel for the server connection, enter:

```
MQSERVER=CHANNEL1/TCP/'mymachine1(1414)'
export MQSERVER
```

4. Restart your application server.

Configuring MQ JMS

Configure the RepConnector environment for MQ JMS.

1. Verify that the lines in the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX that define the IBMMQ_HOME environment variable are not commented out and that they point to the installation location for the IBM MQ JMS. For example:

   On Windows:

   ```
   IBMMQ_HOME=c:\Program Files\IBM\WebSphere MQ
   ```

   On UNIX:

   ```
   IBMMQ_HOME=/opt/mqm
   ```

2. Verify that the lines that define the directory structure for the IBM WebSphere MQ JMS library files, `mq.jar` and `mqbind.jar`, are not commented out and that they are defined correctly for your environment.

   On Windows:

   ```
   REPRA_CLASSPATH=%IBMMQ_HOME%\Java\lib\com.ibm.mq.jar;
   REPRA_CLASSPATH=%IBMMQ_HOME%\Java\lib\com.ibm.mqbind.jar:
   %REPRA_CLASSPATH%
   REPRA_CLASSPATH=%IBMMQ_HOME%\Java\lib\com.ibm.mqjms.jar:
   %REPRA_CLASSPATH%
   REPRA_CLASSPATH=%IBMMQ_HOME%\Java\lib:%REPRA_CLASSPATH%
   CLASSPATH=%CLASSPATH%:%REPRA_CLASSPATH%
   BOOTCLASSPATH=%BOOTCLASSPATH%:%REPRA_CLASSPATH%
   ```

   On UNIX:

   ```
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib/com.ibm.mq.jar
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib/com.ibm.mqbind.jar:
   $REPRA_CLASSPATH
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib/com.ibm.mqjms.jar:
   $REPRA_CLASSPATH
   REPRA_CLASSPATH=$IBMMQ_HOME/java/lib:$REPRA_CLASSPATH
   CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
   BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
   ```

3. If you are connecting to the remote MQ daemon, verify that the MQ Server environment variable is defined correctly in the `repra_env.bat` file on Windows, or the `repra_env.sh` file on UNIX. If the environment variable is not defined correctly, modify it as necessary.

On Windows:

At the command line, where CHANNEL1 is the name you have defined for the channel for the server connection, enter:

```
set MQSERVER=CHANNEL1/TCP/'remotemachine1(1414)'
```

On UNIX:

At the command line, where CHANNEL1 is the name you have defined for the channel for the server connection, enter:

```
MQSERVER=CHANNEL1/TCP/'remotemachine1(1414)'
export MQSERVER
```

4. Restart your application server.

Configuring RepConnector for Your Database

Set up the RepConnector environment so that RepConnector can communicate with the database to send SQL events that it receives from the messaging system. If you are using an Adaptive Server database, no additional steps are required for RepConnector to communicate with the database.

Configuring for an Oracle Database

Configure RepConnector for an Oracle database.

1. Verify that the lines in the `repra_env.bat` file on Windows or `repra_env.sh` file on UNIX that define the CLASSPATH/BOOTCLASSPATH environment variable, are not commented out, and that they point to the installation location for your database environment.

   On Windows:

   ```
   CLASSPATH=d:\oracle\ora92\jdbc\ojdbc14.jar;%CLASSPATH%
   BOOTCLASSPATH=d:\oracle\ora92\jdbc\ojdbc14.jar;%CLASSPATH%
   ```

   On UNIX:

   ```
   CLASSPATH= /work/oracle/ora92/jdbc14.jar;$CLASSPATH
   BOOTCLASSPATH= /work/oracle/ora92/jdbc14.jar;$BOOTCLASSPATH
   ```

2. Restart your application server.

Configuring an Application Server for a Custom Sender Processor or Formatter

Set up the RepConnector environment so that RepConnector can load customized classes for message transformation or message destination routing.

1. Verify that the CLASSPATH variable setting in the `repra_env.bat` file on Windows, or the `repra_env.sh` file on UNIX, includes the full path of the `jar` file containing the customized sender processor or message formatter classes.
   - On Windows:
     ```
     <AppServer_install_dir>\repra\sample\client\sample.jar;
     %CLASSPATH%
     ```
   - On UNIX:
     ```
     <AppServer_install_dir>/repra/sample/client/sample.jar:
     $CLASSPATH
     ```
2. Restart your application server.

See also

- *Customizing the Sender and Formatter Processors* on page 65

Configuring the RepConnector for Replication Server Routing

Customize the RepConnector environment to be used with Replication Server routing. Replication Server routing is used when there is more than one Replication Server between the primary database and RepConnector.

1. From the command prompt, navigate to:

   On Windows:
   ```
   <AppServer_install_dir>\repra\bin
   ```
   On UNIX:
   ```
   <AppServer_install_dir>/repra/bin
   ```
2. In the `repra.properties` file, add:
   ```
   REMOTE_REPSERVER_OPTION=true
   ```
3. If the application server is running, restart it for the new property to take effect.

Configuring RepConnector to Send Carriage Return and Tab

Configure RepConnector to send actual character output to the destination. When there is a carriage return (\n) or tab (\t) in a source text column, RepConnector generates the literal representation "\n" or "\t" in the output stream.

1. From the command prompt, navigate to:

   On Windows:

   ```
   <AppServer_install_dir>\repra\bin
   ```

   On UNIX:

   ```
   <AppServer_install_dir>/repra/bin
   ```

2. In the `repra.properties` file, add:

   ```
   REAL_CHARACTER_OPTION=true
   ```

3. If the application server is running, restart it for the new property to take effect.

Creating and Configuring a New Connection

Add and configure a new RepConnector connection.

Although this task includes all the steps required to add a new connection, some steps are described only in summary. Subsequent tasks give the details of these steps and are referred to from this procedure.

1. Right-click the connection profile, then select **Add a New Connection**.
2. On the Create a New Connection page:
   a) Enter a unique connection name. Do not use dashes or spaces.
   b) Select the inbound type, which is the origin or source of the data.

      The inbound type you select determines what outbound types are available.

      Select one of these inbound sources:
      - REPLICATION – if this connection will accept inbound data from Replication Server. When you select REPLICATION as the inbound type, you can select JMS, TIBCO, IBMMQ, or CUSTOM as the outbound type.
      - JMS – if this connection will accept inbound data from a JMS message queue. When you select JMS as the inbound type, you can select only DATABASE as the outbound type.

- TIBCO – if this connection will accept inbound data from a TIBCO message queue. When you select TIBCO as the nbound type, you can select only DATABASE as the outbound type.
- IBM MQ – if this connection is to accept inbound data from an IBM WebSphere MQ message queue. When you select IBM MQ as the inbound type, you can select only DATABASE as the outbound type.

c) Select the outbound type, which is the target or destination for the data:

- JMS – if this connection is to push outbound data to a JMS message queue. This option is available when you select REPLICATION as the inbound type.
- TIBCO – if this connection is to push outbound data to a TIBCO message queue. This option is available when you select REPLICATION as the inbound type.
- IBM MQ – if this connection will push outbound data to an IBM WebSphere MQ message queue. This option is available when you select REPLICATION as the inbound type.
- CUSTOM – if this connection will push outbound data to a target other than the specific message queues listed in the outbound types field. This option is available when you select REPLICATION as the inbound type.
- DATABASE – if this connection will push outbound data to a database. This option is available when you select a message queue (JMS, TIBCO, or IBMMQ) as the inbound type.

3. Click **Next**.

4. On the General Connection Information page:

a) Verify or modify the Uniform Resource Locator (URL) in the "DBEventStream XSD URL" field.

This is the URL for exposing the XML Schema Definition (XSD) over the network.

The default URL is:

`http://<host_name>:<port_number>/RepraWebApp/dtds/dbeventstream.xsd`

where:

- *<host_name>* is the host name of the target server's `http` listener. The default name is "localhost."
- *<port_number>* is the port number of the target. The default port number is 8000.

If the default information is incorrect, change `localhost` to the host name of the target server's HTTP listener and `8000` to the port number on which the target server's HTTP listener is listening.

For example, if the host name is "mymachine", listening at port 8090, the URL is:

`http://mymachine:8090/RepraWebApp/dtds/dbeventstream.xsd`

b) Select the logging level to use for this connection. Log Level defines the level, or type, of logging in the RepConnector log file, `repra.log`. The level you choose depends

on whether you want to see only error messages or detailed messages in the log. The log file resides in the `<AppServer>\repra\logs` directory on Windows and the `<AppServer>/repra/logs` directory on UNIX. The default logging level is INFO.

Choose:

- FATAL – to see information about severe error events that could lead the application to abort.
- ERROR – to see general errors related to events that could allow the RepConnector environment to continue running, as well as fatal errors.
- INFO – to see informational messages that highlight application progress at a high level, as well as fatal errors and general errors.
- WARNING – to see warnings about potentially harmful situations, as well as fatal errors, general errors, and informational messages.
- DEBUG – to see details about general events that can help during debugging, as well as fatal errors, general errors, informational messages, and warnings.

c) Choose **Autostart Connection**, to start the connection automatically whenever the application server starts. By default, this option is not selected.

In a production environment, you may want to start the connection automatically when the application server starts to do the minimal amount of intervention when restarting servers.

If you are developing and testing your RepConnector connections, you might not want to start the connection automatically when your application server restarts. Once you establish a successful connection, you may want to change the connection properties to automatically start the connection.

Note: When you start a RepConnector connection, Replication Server attempts to connect to it. When you stop a RepConnector connection, and suspend the connection at Replication Server, Replication Server continues to attempt to connect to it, even though the connection is stopped.

---

d) Choose **Custom Plug-in Class** to use a user-defined message formatter with RepConnector rather than the RepConnector default XML formatter.

See *Customizing the Sender and Formatter Processors* on page 65.

e) Choose the encrypted data option if the data from the messaging system is encrypted by Adaptive Server 15.0 encryption.

5. Click **Next**.

The page you see next depends on the inbound type you selected. If you selected:

- REPLICATION – the Replication Server Inbound Information page appears.
- JMS – the JMS Information wizard page appears.
- TIBCO – the TIBCO Messaging Information wizard page appears.
- IBM MQ – the MQ Messaging Information wizard page appears.

6. Select **Ping** to verify that you have entered the information correctly.

   A Pinging Connection Status provides status on whether you have configured the information correctly.

   Note: You must update the `repra_env.bat` file on Windows or the `repra_env.sh` file on UNIX to include the messaging system's libraries in the environment and restart your application server before you can use **ping**.

   Click **OK** to exit the status window.

7. Click **Next**.

   The page you see next depends on the outbound type you selected. If you selected:
   - JMS – message queue or topic, the JMS Information page appears.
   - TIBCO RV or RVCM message queue or topic – the TIBCO Messaging Information page appears.
   - TIBCO AECM message queue – the TIBCO Messaging Information page appears.
   - IBM MQ queues – the MQ Messaging Information page appears.
   - CUSTOM – and selected Custom Plug-in Class in the General Connection Information wizard page, the Plug-in wizard page appears.
   - DATABASE – the Database Connection Information page appears.

8. Click **Next**.

   The Summary page displays the values you entered for the new connection. These values are saved in a properties file called *`connection_name`*`.props`. This file resides in the `repra/conf` directory in your application server installation directory structure.

9. Verify the connection configuration information and select **Finish** to create the new connection.

   To change connection configuration information, select **Back** to return to the specific configuration page you want to modify. When you finish making changes, return to this Summary page and select **Finish**.

See also

- *Get Started with RepConnector Manager* on page 15
- *Customizing the Sender and Formatter Processors* on page 65
- *Configuring Replication Information for REPLICATION Inbound Types* on page 31
- *Configuring JMS Information* on page 32
- *Configuring TIBCO Information* on page 34
- *Configuring IBM MQ Information* on page 35
- *Configuring the RepConnector for Messaging Systems* on page 19
- *Configuring Custom Plug-in Information* on page 36
- *Configuring Database Connection Information* on page 37

Configuring Replication Information for REPLICATION Inbound Types

If you inbound type is REPLICATION, enter the Replication Server inbound information and the Replication Server System Database (RSSD) information.

Entering Replication Server Inbound Information

Define the Data Server Interface (DSI) and the port number that the RepConnector connection will listen on, along with the user ID and password. You can find the required values on the configuration worksheets.

1. On the Replication Server inbound information page, enter the name of the Data Server Interface (DSI) in the DSI Name field. This is the same as the name of the connection you created for RepConnector at Replication Server (3.a).
2. Enter a unique port number for the machine in the DSI Port field. This is the same port number you used when you added an interface entry for RepConnector in the Replication Server interfaces file (3.d)
3. Enter the user name and password for the RepConnector connection. This user name and password must be the same as the user name and password you used when you created the DSI connection for RepConnector at Replication Server (3.e and 3.f).
4. Click **Next**.

   The Replication Server System Database wizard page appears.

See also

- *Configuration Worksheets* on page 93
- *Creating and Configuring a New Connection* on page 27

Entering Replication Server System Database Information

Define the information that RepConnector uses to connect to the Replication Server System Database (RSSD) to gather metadata information for processing events from Replication Server.

1. On the Replication Server System Database Information page, enter the JDBC™ URL string that connects to the RSSD:

   ```
   jdbc:sybase:Tds:<RSSD host machine name>:
      <RSSD port number>/<RSSD database name>
   ```

   where:

   - *jdbc:sybase:Tds* is the URL prefix.
   - *<RSSD host machine name>* is the name of the host machine on which the RSSD is running.
   - *<RSSD port number>* is the port number on which the RSSD is listening.
   - *< RSSD database name>* is the RSSD database name.

   For example:

```
jdbc:sybase:Tds:mymachine:4501/SAMPLE_RS_RSSD
```

2. Enter the user name and password to connect to the RSSD.
3. Select your message grouping preference.
   - Individual – each command in a transaction sent as separate XML message or event.
   - Group – all the commands in a transaction grouped into a single XML message or event.

Note: If you use RepConnector to replicate tables containing large text or image type fields, Sybase recommends that you do not use the Group option. Your system may run out of memory after accumulating only several messages.

See also

- *Creating and Configuring a New Connection* on page 27

Configuring JMS Information

Define the JMS information using the Create Connection wizard if you are connecting to a JMS message queue or topic.

1. Choose the destination type:

Destination Type	Description
Queue	Point-to-point messaging
Topic	Publish-and-subscribe messaging.

2. Enter the JMS provider URL, which is the host name and port number that is used to connect to the JMS Server.

JMS Provider	JMS Provider URL	Where

JBoss HornetQ JMS	`jnp://<host_name>:<port_number>`
	For example:
	`jnp://localhost:1099`
		• *host_name* – is the name of the machine on which the server is running.
		• *port_number* – is the port number at which the server is listening.

WebLogic Server JMS (protocol type must be “t3”, which is the WebLogic multitier JDBC driver)	`t3://<host_name>:<port_number>`
	For example:
	`t3://localhost:7001`	

JMS Provider	JMS Provider URL	Where

TIBCO JMS or SonicMQ JMS	`tcp://<host name>:<port number>`
	For example:
	`tcp://localhost:7222`	

3. Enter or select the class name of the specific JMS provider's initial naming context factory.

JMS Provider	Class Name

JBoss HornetQ JMS	For JBoss 6:
	`org.jnp.interfaces.NamingContextFactory`
	For JBoss 7:
	`org.jboss.as.naming.InitialContextFactory`

WebLogic Server JMS	`weblogic.jndi.WLInitialContextFactory`
TIBCO JMS	`com.tibco.tibjms.naming.` `TibjmsInitialContextFactory`

SonicMQ JMS	If the destination type is a queue:
	`progress.message.jclient.QueueConnectionFactory`
	If the destination type is a topic:
	`progress.message.jclient.TopicConnectionFactory`

4. Enter or select the object that is administered by the connection factory.

JMS Provider	Connection Factory Name
JBoss HornetQ JMS	`java:/ConnectionFactory`
WebLogic Server JMS	`weblogic.jms.ConnectionFactory`

TIBCO JMS	If the destination type is a queue:
	`com.tibco.tibjms.TibjmsQueueConnectionFactory`
	If the destination type is a topic:
	`com.tibco.tibjms.TibjmsTopicConnectionFactory`
	Make sure that this connection-factory-administered object name exists in your TIBCO JMS Server, or change the value here to match the object you created in TIBCO JMS Server.

JMS Provider	Connection Factory Name

SonicMQ JMS	If the destination type is a queue:
	`progress.message.jclient.QueueConnectionFactory`
	If the destination type is a topic:
	`progress.message.jclient.TopicConnectionFactory`

5. Enter the name of the destination; for example, if the destination is a JMS queue, enter:
   ```
   JMS_Queue
   ```
6. Enter the user name and password for the queue or topic, for example, enter `jagadmin` with no password.
7. If the destination type is a topic, enter the name of one or more durable subscribers in the Topic Subscribers field, separated by commas, with no spaces in between.

   Durable subscribers are subscribers who are interested in receiving messages from the selected published topic. For example, enter:
   ```
   JMSTSub1,JMSTSub2,JMSSub1
   ```
8. If you are routing events from messaging to database, enter name of the destination in the Status Destination field.

   The status destination queue or topic you define is used for a client application to listen for an error message (if any) that may result in the event sent to the database.

See also

- *Creating and Configuring a New Connection* on page 27

Configuring TIBCO Information

If you are using a TIBCO messaging system, define the TIBCO information using the Create Connection wizard.

1. In the TIBCO Message Type field, choose:
   - RV – to configure a TIBCO Rendezvous-reliable message.
   - RVCM – to configure a TIBCO Rendezvous Certified Messaging (RVCM) message.
   - AECM – to configure a TIBCO Rendezvous Active Enterprise Wired Format messaging message. Skip to step 3.
2. If you selected RV or RVCM:
   a. In Service Name, enter the name of the RV and RVCM transport. The service name can be either a string value or a port number. By default, the value is 7500.
   b. In Network, enter the name of the host name or IP address where the TIBCO Rendezvous daemon is running. For example, enter `job1-srvr`.
   c. In Daemon, enter the TIBCO Rendezvous daemon value:
      ```
      protocol:hostname:port
      ```
      For example, enter `tcp:my_machine:7500`.

The default value is tcp:7500, which defaults to “localhost” on port 7500.

If your TIBCO Rendezvous daemon is running on a machine other than the one on which RepConnector is running, you must specify the host name; for example:

```
tcp:mymachine:7500
```

d. Enter the name of the subject or destination at which the client application is listening. For example, enter `sample.subject`.

You can specify more than one subject name, separated by commas.

To preregister listeners, enter the main subject, followed by a comma, followed by the name and subject pairs of the listeners you are preregistering, separated by a colon:

```
SAMPLE.REPC15.EVENT,cmNameA:cmSubA,cmNameB:cmSubB
```

In this example, SAMPLE.REPC15.EVENT is the subject to publish to and cmNameA:cmSubA,cmNameB:cmSubB are the name and subject pairs you are preregistering.

e. If you are using an RVCM message type, enter the certified messaging names in the CM Names field. For example, enter `SAMPLE.CM1`.

You can specify more than one subject name, separated by commas.

f. If you are using an RVCM message type, enter, in the CM Duration field, the number of seconds that TIBCO message system should store unread messages in the `cmledger`.

The default value is 0, unread messages are never removed from the `cmledger` file.

For example, enter 600 to have the messaging system keep unread messages for 10 minutes.

3. If you selected AECM as the TIBCO message type:

- Enter the name of the AE configuration file you are using. Or, click **Browse** to search for this file.
- If you want RepConnector to use your customized message generator, enter the class name for your customized message generator in the AE Message Generator field. For example, enter `sample.MyMsgGenerator`

4. If you are routing events from a messaging system to a database, enter the status destination. The status destination queue or topic is where the client application listens for any error messages that may result in the event being sent to the database.

See also

- *Creating and Configuring a New Connection* on page 27

Configuring IBM MQ Information

Define IBM MQ information.

1. Choose an MQ message type:

- MQ for IBM MQ Messaging System

- MQJMS for IBM MQ JMS Messaging System

2. Choose:
   - Local Server, if you are running the MQ server daemon (IBM MQ server) on your local machine.
   - Local Client, if you are running the MQ client daemon (IBM MQ client) on your local machine.

   Select the encoding type for the message:
   - Default – to use standard character encoding.
   - UTF – to use the UTF character encoding.
3. Enter the host name where the MQ Server daemon is running. For example, enter `mqbiz2-pc`.
4. Enter the channel name for the IBM MQ server connection.

   If you have selected MQ JMS as the MQ message type, enter the port number in the channel field. For example, enter `1414`.
5. In the Queue Manager/Factory field, enter the name of the IBM MQ queue manager. For example, enter `MQBiz2QM`.
6. In the Queue Name field, enter the name of the IBM MQ destination. For example, enter `MQBiz2Queue`.
7. Enter the user name and password to connect to IBM MQ Server.

   Note: Verify that this user name and password combination has permission to connect to the queue manager defined in the Queue Manager/Factory field and to the destination name defined in Queue Name field.
8. If you are routing events from a messaging system to a database, enter the error queue name in the Status Destination field.

   The status destination queue is used by client applications to catch error messages, which may stop applications from sending events to the database.

See also

- *Creating and Configuring a New Connection* on page 27

Configuring Custom Plug-in Information

If you are using a custom message formatter or custom message sender processor, define the plug-in class information using the Create Connection wizard.

1. Select **Customized Plug-in Class** in the General Connection Information wizard to load a custom formatter, and enter the class name for your custom message formatter in the Message Formatter Plug-in Class field. For example, enter:

```
sample.MyMessageFormatter
```

If you have a property file associated with this custom message formatter, enter it, along with the full path name in the Message Formatter Properties File field. For example, enter:

```
\classes\myclasses\MyMessageFormatter.prop
```

2. Select CUSTOM as your outbound type to load a custom sender, and enter the class name for your custom sender in the Sender Processor Plug-in Class field. For example, enter:

   ```
   sample.FileSender
   ```

   If you have a property file associated with this custom sender, enter it along with the full path name in the Sender Processer Properties File field. For example, enter:

   ```
   \classes\myclasses\MyCustomFileSender.prop
   ```

See also

- *Creating and Configuring a New Connection* on page 27

Configuring Database Connection Information

If your outbound connection is to a database, define the database connection information using the Create Connection wizard.

1. Enter the JDBC URL information to connect to the database.

   For example, to connect to Adaptive Server, enter:

   ```
   jdbc:sybase:Tds:testmachine:5000
   ```

   where "testmachine" is the name of the machine where the data server is running and 5000 is the port number where the data server is listening.
2. Enter the name of the JDBC driver class to connect to the database.

   For example, the JDBC driver to connect to Adaptive Server is:

   ```
   com.sybase.jdbc3.jdbc.SybDriver
   ```
3. Enter the user name and password to connect to the database.

See also

- *Configuring RepConnector for Your Database* on page 25
- *Creating and Configuring a New Connection* on page 27

Managing a RepConnector Connection

Manage a RepConnector connection using the RepConnector Manager or the command line utility, **ratool**. Make sure that you have already connected to the RepConnector instance.

See also

- *Get Started with RepConnector Manager* on page 15
- *ratool Utility* on page 41

Starting a Connection

Strart a RepConnector connection.

1. Right-click the connection and select **Start Connection**.
2. Once the status indicates that the connection has successfully started, click **OK**.

Stopping a Connection

Stop a RepConnector connection.

1. Right-click the connection and select **Stop Connection**.
2. Click:

Yes	To stop the connection.
No	To cancel the operation.

3. When the status window indicates that the connection has successfully stopped, click **OK**.

Refreshing a Connection

Refresh a RepConnector connection.

1. Right-click the connection and select **Refresh Connection**.
2. When the connection has finished refreshing, click **OK**.

Renaming a Connection

Rename a RepConnector connection.

Prerequisites

Stop the connection.

Task

1. Right-click the connection and select **Rename Connection**.
2. Enter the new connection name.
3. Click **OK**.

Deleting a Connection

Delete a RepConnector connection.

Prerequisites

Stop the connection.

Task

1. Right-click the connection and select **Delete Connection**.
2. Click **OK**. Confirm the deletion, or click **Cancel** to cancel the delete operation.

Copying a Connection (Save As)

Create a new RepConnector connection by copying the connection properties of an existing connection.

1. Right-click the connection and select **Save As**.
2. Enter the new connection name.
3. Click **OK**.

Validating a Connection

Validate both the inbound and outbound configuration properties.

1. Right-click the connection and select **Validate Connection**.
2. Click **OK**.

Viewing or Modifying Properties for an Existing Connection

View or modify connection properties of an existing connection. If the connection is already running, select **Refresh Connection** to reload the connection properties.

1. Right-click the connection and select **Properties**.

   The left pane of the Properties window contains a tree structure with four categories of connection property information. Select:

   - General Properties – to view general information about the connection.
   - Inbound Type Properties – to view inbound configuration information.
   - Outbound Type Properties – to view outbound configuration information.
   - User-Defined Plug-in Properties – if this connection contains a customized plug-in.
2. Modify the values in the right pane.
3. (Optional) Click **Restore Defaults** to restore the previous values.

4. Click **Apply** to save the values.
5. Click **OK** to save the values to the connection property repository.

Viewing Connection Log Information

View RepConnector connection log information.

1. Right-click the connection and select **View Log**.
2. When you have finished viewing the log information, close the window.

To view any updates to the connection log since you last opened the View Connection Log window, exit the current view log window, then right-click the connection and select **View Log**.

Viewing the Runtime Log Information

View the RepConnector runtime log information.

1. Right-click the connection profile and select **View Log**.
2. When you have finished viewing the runtime log information, close the window.

To view any updates to the runtime log since you last opened the view log window, exit the current View Runtime Log window, then right-click the profile and select **View Log**.

Refreshing the Connection View

Refresh the RepConnector connection view. If you have added a new connection from another RepConnector Manager or through the command line tool, select **Refresh** to see newly added connections for the RepConnector runtime instance.

1. Right-click the connection profile.
2. Select **Refresh View**.

ratool Utility

Sybase recommends that you use RepConnector Manager to administer and configure connections. However, if you are performing batch processing, **ratool**, a command line utility, can be a useful alternative.

You can use **ratool** to start, stop, refresh, add, delete, and validate a connection; list all connections, and display status for the connection.

ratool is located in `RepConnector_install_dir/repra/bin` on Windows and `RepConnector_install_dir\repra\bin` on UNIX. To use **ratool**, either add `repra/bin` on Windows or `repra\bin` on UNIX to your path, or access it directly from this directory.

ratool

If you are performing batch processing, **ratool** is an alternative to using RepConnector Manager to administer and configure connections.

Note: Command options are case-sensitive.

Syntax

```
ratool [-host hostname] [-port portnumber] [-user <username>]
[-password <password>] [-help] [-loglevel <loglevel_type>]
[<Command_option>]
```

Options	Definitions
`-host` *<hostname>*	Identifies the name of the host on which the RepConnector runtime is running. The default is “localhost”.
`-port` *<portnumber>*	The port number on which the RepConnector runtime instance is listening. The default value is 8000.
`-user` *<username>*	The RepConnector administrator login user ID. The default is re-praadmin.
`-password`*<password>*	The password for the RepConnector administrator login. By default, there is no administrator password.
`-help`	Displays information about all **ratool** commands.

Options	Definitions
`help` *<command_option>*	Displays the help information for a specific **ratool** *command_option*. Do not include "-" before the help command flag when you display help information for a command option.
`-logfile` *<file_name>*	Sends the logging information for **ratool** to a specified file.
`-loglevel` *<loglevel_type>*	Determines the level of logging information to show. Valid values are: • FATAL • ERROR • WARNING • INFO • DEBUG
Command_options	
`-copy` *<conn_name> <new_conn_name>*	Copies a connection name to a new connection name.
`-delete` *<conn_name>*	Deletes a connection.
`-getLogInfo` *<connName>*`[-file` *<logFile>*`]`	Displays the logging information for a specified connection. If you specify **-file**, **-getLogInfo** writes the logging information to a file specified by *logFile*.
`-getProperty` *<connName>*`[-file` *<propfile>*`]`	Displays the logging information for a specified connection. If you specify **-file**, **-getProperty** writes the logging information to a file specified by *logFile*.
`-import` *<conn_name> <conn_prop_file>* **[-override]**	Adds a new connection. If you specify **-override**, **-import** updates the connection property information for an existing connection.
`-list`	Lists all known connections.

Options	Definitions
`-ping` *<conn_name>* *<pingType>*	Verifies the connection is configured correctly. Valid values are: • ALL • IBMMQ • TIBCO • JMS • DATABASE REPLICATION • INBOUND • OUTBOUND
`-refresh` *<conn_name>*	Refreshes a specific connection.
`-refreshAll`	Refreshes all connections.
`-rename` *<conn_name>* *<new_conn_name>*	Renames a connection.
`-start` *<conn_name>*	Starts a specific connection.
`-startAll`	Starts all connections
`-status` *<conn_name>*	Lists connection status. If no connection name is specified, this command option gives the status of all known connections. Valid values are: • STARTING • RUNNING • STOP
`-stop` *<conn_name>*	Stops a specific connection.
`-stopAll`	Stops all running connections.
`-validate` *<conn_prop_file>*|| *<conn_name>*	Validates a new connection profile or an existing connection.

-copy

Copies the connection name.

Syntax

```
ratool -copy <src_connection_name> <dest_connection_name>
```

Parameters

- ***src_connection_name* –** the connection to copy.
- ***dest_connection_name* –** the connection you are creating from the source connection.

Examples

- **Example 1 –** copies the connection named RepToJMS to a new connection named RepToJMS2:

  ```
  ratool -copy RepToJMS NewRepToJMS
  ```

Usage

If the destination connection already exists, you see:

```
RaCommand[ERROR]: Copy connection failed. Error
Message: com.sybase.connector.repra.RaException:
java.lang.Exception: The destination connection already
exists.
```

-delete

Deletes a specified connection.

Syntax

```
ratool -delete <connection_name>
```

Parameters

- ***connection_name* –** the connection you want to delete.

Examples

- **Example 1 –** deletes a connection RepToJMS:

  ```
  ratool -delete RepToJMS
  ```

Usage

If the connection is running, this option returns:

```
RaCommand[Error]: Delete Connection failed. Error
Message: com.sybase.repra.util.RaException:
java.lang.Exception: The connection cannot be deleted
since it is currently in a STARTING or RUNNING state.
```

-getLogInfo

Retrieves connection log information for a specified connection.

Syntax

```
ratool -getLogInfo <connection_name> [-file <log_file>]
```

Parameters

- ***connection_name* –** the connection for which you want log information.
- ***log_file* –** the log file to which you are sending the connection log information.

Examples

- **Example 1 –** Gets the connection log information for connection RepToJMS:

  ```
  ratool -getLogInfo RepToJMS
  ```

- **Example 2 –** Gets the connection log information for connection RepToJMS and sends it to `RepToJMS.log`:

  ```
  ratool -getLogInfo RepToJMS -file RepToJMS.log
  ```

- **Example 3 –** Gets the connection log information for connection RepToJMS and sends it to the default log file (`defaultRepToJMS.log`):

  ```
  ratool -getLogInfo RepToJMS -file
  ```

Usage

By default, log information is sent to a standard output screen. If you specify **-file**, log information is sent to the specified file. If you specify **-flag** without a file name, log information is sent to a default file, `default<connection_name>.log`.

-getProperty

Retrieves connection property information for a connection.

Syntax

```
ratool -getProperty <connection_name> [-file <props_file>]
```

Parameters

- ***connection_name* –** the connection for which you want log information.
- ***props_file* –** the file to which you are sending the connection property information.

Examples

- **Example 1 –** Gets the connection property information for connection RepToJMS:

  ```
  ratool -getProperty RepToJMS
  ```

- **Example 2 –** Gets the connection property information for connection RepToJMS and sends it to `RepToJMS.props`:

  ```
  ratool -getProperty RepToJMS -file RepToJMS.props
  ```

- **Example 3 –** Gets the connection property information for connection RepToJMS and sends it to the default connection file (`defaultRepToJMS.props`):

  ```
  ratool -getProperty RepToJMS -file
  ```

Usage

The information is returned in a *name/value* pair. By default, the information is sent to standard output (`stdout`). If you specify `-file`, the connection property information is sent to the specified file name. If you specify **-flag** without a corresponding file name, the log information is sent to a default file, `default<connection_name>.props`.

-import

Imports connection properties from an existing file to create a new connection or update an existing connection.

Syntax

```
ratool -import <connection_name> <connection_prop_filename> [-
override]
```

Parameters

- ***connection_name* –** the connection to import.
- ***connection_prop_filename* –** the file that contains the properties to import.
- **-override –** overrides the existing connection information. If you do not specify `-override` and there is an existing connection, you see:

```
RaCommand[ERROR] Import connection failed. Error
message: com.sybase.connector.repra.RaException:
java.lang.Exception: The existing connection cannot be
overriden
```

Examples

- **Example 1 –** Adds a new connection using the properties in `RepToJMS.props`:

```
ratool -import RepToJMS d:\repraconf\RepToJMS.props
```

- **Example 2 –** Updates an existing connection using the properties in `RepToJMS.props`:

```
ratool -import RepToJMS d:\repraconf\RepToJMS.props -override
```

Usage

See the sample configuration property files in the RepConnector `sample/conf` directory for information about property names and values.

-list

Lists all known connections.

Syntax

```
ratool -list
```

Examples

- **Example 1 –** lists all connections:

```
ratool -list
```

-ping

Verifies that the connection is configured successfully by pinging the connection.

Syntax

```
ratool -ping <connection_name> <ping_type>
```

Parameters

- ***connection_name* –** the connection for which you are verifying the configuration information.
- ***ping_type* –** the type of connection you are pinging to verify configuration information. Valid values for *ping_type* are:
  - ALL – inbound and outbound routes.
  - IBMMQ – IBM WebSphere MQ messaging system.
  - TIBCO – TIBCO messaging system.
  - JMS – JMS messaging system.
  - DATABASE – server in which the database resides.
  - REPLICATION – Replication Server System Database (RSSD).
  - INBOUND – inbound source configuration (for example, server or messaging system).
  - OUTBOUND – outbound destination configuration (for example, server or messaging system).

Examples

- **Example 1 –** verifies both the inbound and outbound configuration for the connection RepToJMS by pinging both inbound and outbound routes:

```
ratool -ping RepToJMS
```

```
ratool -ping RepToJMS ALL
```

- **Example 2 –** verifies the inbound configuration for connection RepToJMS by pinging the inbound device (for example, server or messaging system):

```
ratool -ping RepToJMS INBOUND
```

- **Example 3 –** verifies the outbound configuration for connection to RepToJMS by pinging the outbound destination device (for example, server or messaging system):

```
ratool -ping RepToJMS OUTBOUND
```

- **Example 4 –** verifies the replication configuration for a connection to RepToJMS by pinging Replication Server:

```
ratool -ping RepToJMS REPLICATION
```

- **Example 5 –** verifies the JMS configuration for connection to RepToJMS by pinging the JMS messaging system:

```
ratool -ping RepToJMS JMS
```

Usage

If you do not specify *ping_type*, the value defaults to ALL.

-refresh

Refreshes a specified connection.

Syntax

```
ratool -refresh <connection_name>
```

Parameters

- ***connection_name* –** the connection to refresh.

Examples

- **Example 1 –** refreshes the connection RepToJMS:

  ```
  ratool -refresh RepToJMS
  ```

Usage

This option reloads the connection properties if they have changed, and restarts the connection. **refresh** applies only to running connections. If the connection is not running, you see a warning message.

-refreshAll

Refreshes all running connections.

Syntax

```
ratool -refreshAll
```

Examples

- **Example 1 –** refreshes all the running connections.

  ```
  ratool -refreshAll
  ```

Usage

This option reloads the connection properties if they have changed, and restarts the connection. **refreshAll** applies only to running connections.

-rename

Renames a connection.

Syntax

```
ratool -rename <old_connection_name> <new_connection_name>
```

Parameters

- ***old_connection_name* –** the connection to rename.
- ***new_connection_name* –** the new name for the connection.

Examples

- **Example 1 –** To rename connection RepToJMS to RepToJMS2:

  ```
  ratool -rename RepToJMS NewRepToJMS
  ```

Usage

If you attempt to rename a connection that is running, or the new connection name already exists, you see an error message.

-start

Starts a specified connection.

Syntax

```
ratool -start <connection_name>
```

Parameters

- ***connection_name* –** the connection to start.

Examples

- **Example 1 –** Starts a connection called RepToJMS:

  ```
  ratool -start RepToJMS
  ```

Usage

If the connection is already running, **ratool** returns a warning message.

-startAll

Starts all the connections.

Syntax

```
ratool -startAll
```

Examples

- **Example 1 –** starts all known connections.

  ```
  ratool -startAll
  ```

-status

Gets the status of a specific connection.

Syntax

```
ratool -status [ <connection_name> ]
```

Parameters

- ***connection_name* –** the connection you want to status for.

Examples

- **Example 1 –** gets the status of RepToJMS:

  ```
  ratool -status RepToJMS
  ```

- **Example 2 –** gets the status of all configured RepConnector connections for a RepConnector running on “localhost”, listening on port 8000, and connecting as user “repraadmin” with no password:

  ```
  ratool -status
  ```

- **Example 3 –** displays debug logging information while running **ratool**:

  ```
  ratool -loglevel DEBUG -status
  ```

  By default, if you do not specify **-logfile**, logging information is sent to standard output.

- **Example 4 –** sends debug logging information to a log file while running **ratool**:

  ```
  ratool -loglevel DEBUG -logfile ratool.log -status
  ```

- **Example 5 –** if you have configured your application server to listen on a different port, for example, 8888, this command shows the status of all configured RepConnector connections:

```
ratool -port 8888 -status
```

- **Example 6 –** if you have changed the default user name “repraadmin” with no password to the new user “newuser” with password “newpassword” for connecting to RepConnector running on “machine1” and port 8888, this command gets the status of the RepConnector connection:

```
ratool -host machine1 -port 8888 -user  newuser -password
newpassword
-status
```

- **Example 7 –** to connect to RepConnector that is running on remote “machine1” listening on default port 8000, connecting as the default user or password, issue one of these commands:

```
ratool -host machine1 -status
ratool -host machine1 -port 8000 -status
ratool -host machine1 -port 8000 -user repraadmin -status
```

- **Example 8 –** if you are connecting to RepConnector running on WebLogic Servers default 7001 port, you can use:

```
ratool -port 7001 -user repraadmin -status
```

Usage

If you do not specify a connection name, the status of all connections displays. Status values for the connection are:

- RUNNING – the connection is running.
- STOP – the connection is not running.

-stop

Stops a specified connection.

Syntax

```
ratool -stop <connection_name>
```

Parameters

- ***connection_name* –** the connection to stop.

Examples

- **Example 1 –** stops connection RepToJMS:

```
ratool -stop RepToJMS
```

Usage

If the connection is already stopped, this option returns a warning message.

-stopAll

Stops all running connections.

Syntax

```
ratool -stopAll
```

Examples

- **Example 1 –** stops all known connections:

```
ratool -stopAll
```

Usage

If no connections are running, this option returns a warning message.

Message Generator for TIBCO AECM Customization

RepConnector supports the TIBCO Active Enterprise wire format feature, which lets you to customize and generate a TIBCO Active Enterprise message. The TIBCO adapter then uses the customized message generator to send the message to the TIBCO RV bus.

Learn the basic implementation of the base class, the structure of a customized class to extend the base class, and the APIs defined in the base class that retrieve the metadata and data from the message source. RepConnector loads the TIBCO AECM client, which loads the SDK repository information. The user exit is where you can create your own Java implementation to customize a wire-formatted message.

Configuring Properties for RepConnector

Configure RepConnector connection properties to use the TIBCO AECM feature along with the message generator customization class, and the Active Enterprise properties required to connect to the TIBCO SDK repository and to generate the customized wire-format message.

Connection Configuration

You can use the TIBCO Active Enterprise feature and the customized message generator with configuration parameters for TIBCO Active Enterprise.

Table 1. Parameters for TIBCO Active Enterprise

Property Name	Description
Inbound Type	The inbound type must be set to REPLICATION: `Inbound Type=REPLICATION`
Outbound Type	The type of sender the client processor uses for sending messages. In this case, TIBCO: `Outbound Type=TIBCO`
TIBCO Message Type	The transport message type for TIBCO must be AECM.

Property Name	Description
AE Configuration File	Active Enterprise configuration properties. Enter the full path name to where the property file is located. This property file contains connection information to the SDK repository, as well as the properties required for customizing the message. For example: `AppConfig=C:/Sybase/appserver/repra/conf/ae.props`
AE Message Generator	The Customized Message Generator class name, and Active Enterprise-specific property. For example: `MsgGenerator=sample.MyMsgGenerator`

Property File with the Active Enterprise Connection/Customization (ae.props)

Use the required connection properties to connect to the SDK repository. You can customize additional properties for your message generator, such as the schema class name. Use the full path of this file as the value of the **AE Configuration file** property of the connection.

Table 2. Properties for Connection to SDK

Property Name	Description
application_name	The application name of your SDK adapter. For example: `application_name=simpleSDK_adapter`
application_version	The application version of your SDK adapter. For example: `application_version=1.0`
application_info	The application description of your SDK adapter. For example: `application_info=fileadapterinfo`
config_URL	The location of your application inside the SDK repository. For example: `config_URL=/tibco/private/adapter/sdkTest_Adapters/simpleSDK_adapter`
remote_repository	The location (full path) to the SDK repository. For example: `remote_repository=d:/Sybase/appserver/repra/conf/sampleSDK.dat`
data_publish_name	The name of the publisher that this application is using. For example: `data_publish_name=myPub`

Property Name	Description
pub_subject	The subject name that the publisher is going to publish on. For example: `pub_subject=repraTest.subject1`
pre_registered_subscribers	The list of subscribers to preregister, separated by a comma. For example: `pre_registered_subscribers=myCmListener,myCmListener2`
command_args	(Must be entered as a single line) The command line argument to initialize the SDK application. For example: `command_args=-system:repourl ../repra/conf/sampleSDK.dat-system:configurl/tibco/private/adapter/sdkTest_Adapters/simpleSDK_adapter`

Base Class APIs

You can use base class APIs to build a custom TIBCO AECM message generator.

By default, the message generator base class converts a RepEvent object to a well-formed M-tree object in a simple format. The class places the XML text stream into the data field of the Active Enterprise message and adds it to the M-tree node.

Customizing TIBCO AECM Message Generator

Generate a customized message generator to create a well-formed M-tree object of a certain wire format. To do this, extend the base class MsgGenerator and implement your own **createMInstance** method.

The parameter to **createMInstance()** is a RepEvent object. The following APIs defined in the base class allow you to retrieve specific information for your customization. Extend this base class to customize your message generator.

There are public methods to help you retrieve the data object information, Active Enterprise customized user properties, and the MClassRegistry.

These are the required methods for the customized message generator:

```
public class MsgGenerator implements WireFormatGenerator
{
    /** This method returns a well-formed MTree of a certain
    * WireFormat. You will need to extend this method
    * to customized your MsgGenerator.
    */
    public MTree createMInstance(Object repEvent) throws Exception

    /* Other APIs provided for retrieving information from
    * the RepEvent Object provided in the next section.
```

```
    */
...
}
```

The extending class must have a public constructor with no input argument. For example:

```
import com.sybase.connector.repra.tibrv.MsgGenerator;
import com.sybase.connector.repra.util.*;
public class MyMsgGenerator extends MsgGenerator
{
...
// This is the default constructor
public MyMsgGenerator()
{
}
...
}
```

To customize the message format, use the extending class implementation of the **createMInstance()** method. For example:

```
public MTree createMInstance(Object repmsg)
throws Exception
{
    MTree mTree = new MTree("msg");
    ...
    // do something to build the message MTree
    return mTree;
}
```

APIs for a Customized, Wire-Format Message Generator

The APIs (embedded in the base class MsgGenerator) that you can use to build a custom TIBCO AECM Message Generator retrieve information from the RepEvent object.

MClassRegistry getClassRegistry()
Returns the current MClassRegistry object.

String getOwner(int elementAt) throws Exception
Retrieves the owner name of the replication event, when extending the base message generator `com.sybase.connector.repra.tibrv.MsgGenerator`.

Example:

```
          System.out.println("Owner of the table : "+getOwner(0));
```

String getProperty(String key)
Returns a string value with the given key from the properties file defined as the AE Configuration file of the connection configuration. Returns a null value if the key is not found.

String getProperty(String key, String defValue)
Returns a String value with the given key from the properties file defined AE Configuration file of the connection configuration. Returns the defValue if the key is not found.

setProperties(Properties props)
Sets the current properties with the input Properties object.

Properties getProperties()
Returns the current properties.

M-Tree createMInstance(Object repmsg) throws Exception
Builds the M-tree for the TIBAECM client and returns it. The MPublisher sends out this M-Tree object to MSubscribers.

APIs That Retrieve Information from the Source Event

The base class MsgGenerator also provides additional APIs to retrieve the metadata and replication data from the replication events.

See also

- *DBEventParserFactory Utility* on page 75

Configuration and Default Wire-Formatted Message Generator Usage

Configure connections and use the default wire-formatted message generator.

An example of configuring connections:

```
Inbound Type = REPLICATION
Outbound Type = TIBCO
TIBCO Message Type = AECM
AE Configuration File = d:\sybase\appserver\conf\ae.props
AE Message Generator =
```

An example of SDK application configuration with `d:\sybase\appserver\repra\conf\ae.props`:

```
application_name=simpleSDK_adapte
application_version=1.0
application_info=fileadapterinfo
config_URL=
/tibco/private/adapter/sdkTest_Adapters/simple
SDK_adapter
remote_repository=
      Sybase/appserver/repra/conf/sampleSDK.dat
data_publish_name=myPub
pub_subject=repraTest.subject1
```

```
pre_registered_subscribers=myCmListener,myCmListener2
```

Example output:

By default, the message format is an XML text-stream representation of the RepEvent. In a Tibrv Listener, the format is:

```
message = {
^pfmt^ = 10
^ver^ = 30
^type^ =1
^encoding^ =1
^tracking^={^id^="5rRX7g5jVWROok8EujzzwB3Uzzw"}
^data^={
    data={RepEvent="<?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE dbStream SYSTEM
            'http://yjeongw2k:8000/repra/dtds/dbeventstream.dtd'>
      <dbStream environment="repraJMS2.repdb">
            <tran
eventId="102:000000000000c8d500007fe7003900007fe70036000093bd00bf1c
0c0000000000010001">
              <insert schema="REP4">
                 <values>
                   <cell name="repId" type="INT">2</cell>
                   <cell name="repName" type="VARCHAR">name 2</cell>
                   <cell name="repCode" type="VARCHAR">code 2</cell>
                 </values>
              </insert>
            </tran>
      </dbStream>"}
}
}
```

This message has been formatted for readability.

Configuration and Customized Wire-Formatted Message Generator Usage

The summarized overview of the different components from the back-end server, Adaptive Server® Enterprise, and Replication Server, as well as the information in configuring the RepConnector to deliver a TIBCO AE message to a TIBCO message bus are described.

Note: This discussion assumes knowledge of the back-end server and the SDK design.

There is a table called `REP4` in a database called `repdb` that resides in Adaptive Server Enterprise. This table contains two columns called `repName` and `repCode`.

Example

```
Inbound Type=REPLICATION
Outbound Type=TIBCO
TIBCO Message Type=AECM
AEConfiguration=d:\sybase\appserver\repra\conf\ae.props
AE Message Generator =MyMsgGenerator
```

SDK Application Configuration Sample

The SDK application configuration file contains information that is required to connect to the SDK repository, and user-defined parameters that can be used by the customized message generator.

Here is an example of the contents of the SDK application configuration file:

```
application_name=simpleSDK_adapter
application_version=1.0
application_info=fileadapterinfo
config_URL=/tibco/private/adapter/sdkTest_Adapters/
simpleSDK_adapter
remote_repository=F:/appserver/repra/conf/sampleSDK.dat
data_publish_name=myPub
pub_subject=repraTest.subject1
pre_registered_subscribers=myCmListener,myCmListener2
command_args=-system:repourl ../repra/conf/sampleSDK.dat -
system:configurl /tibco
/private/adapter/sdkTest_Adapters/simpleSDK_adapter
context_schema_class=SybContext
native_schema_class=SybNATIVEMSG
commonmsg_schema_class=SybCommonMSG_UDS
ContextKeys=repName,repCode
```

The example defines three schema class names and the context keys that map to the definition in the customized SDK repository design.

Code Sample

See the `%REPRA_HOME%\sample\client\MyMsgGenerator.java` file on Windows or `$REPRA_HOME/sample/client/MyMsgGenerator.java` file on UNIX for an example for the customized AE message generator class.

Compiling the Customized Message Generator

Compile the generated customized message generator.

1. Change to the location of your message generator:
   - On Windows:
     ```
     cd C:\work\custom
     ```
   - On UNIX:
     ```
     cd /work/custom
     ```
2. Use the Java compiler to define the `-classpath` parameter with the required libraries to compile the customized class. For example:
   - On Windows:
     ```
     md customclasses <enter>
     ```
     ```
     c:\jdk7\bin\javac -classpath .; <AppServer_install_dir>\repra
     \lib\repraconn.jar
     ```

```
-d customclasses com\mycompany\MyMsgGenerator.java
```

- On UNIX:

```
mkdir customclasses<enter>
```

```
/usr/jdk7/bin/javac -classpath
.:<AppServer_install_dir>/repra/lib/repraconn.jar
<enter>
-d customclasses
com/mycompany/MyMsgGenerator.java
```

3. If the compilation command finishes without any error messages, go to the `customclasses` directory, and verify that `MyMsgGenerator.class` exists in `com\mycompany` on Windows, or `com/mycompany` on UNIX.
4. If `MyMsgGenerator.class` is not in `com\mycompany`, or if the compilation finished with errors, review the design.

Building the Runtime Environment for the Customized Message Generator

Sybase recommends that you build a `jar` file for customization. Otherwise, you can use the directory path to set up your environment.

1. Change to the `customclasses` directory.
   - On Windows:

     ```
     cd C:\work\custom\customclasses
     ```
   - On UNIX:

     ```
     cd /work/custom/customclasses
     ```
2. Build the `jar` file:
   - On Windows:

     ```
     C:\jdk141\bin\jar -cf mycustom.jar com
     ```
   - On UNIX:

     ```
     /usr/jdk141/bin/jar -cf mycustom.jar com
     ```
3. Add the path to `mycustom.jar` to your environment.
   a. Shut down the application server if it is running.
   b. Modify `<RepConnector_home_dir>\bin\repra_env.bat` (Windows) or `<RepConnector_home_dir>/bin/repra_env.sh` (UNIX) to add the full path of `mycustom.jar` or the `customclasses` directory to the end of the CLASSPATH definition.
      - On Windows, either:
        - `set CLASSPATH=C:\work\custom\customclasses;%CLASSPATH%`

          or:
        - `set CLASSPATH=C:\work\custom\customclasses\mycustom.jar; %CLASSPATH%`
      - On UNIX, either:

- `CLASSPATH=/work/custom/customclasses:$CLASSPATH`

  or:
- `CLASSPATH=/work/custom/customclasses/mycustom.jar; $CLASSPATH`

c. Start the application server to activate the environment changes.

Example Output for TIBRV Listener and Active Enterprise Wire Format

An example output for TIBRV listener and Active Enterprise wire-formatted message generator.

The example messages have been formatted for readability.

```
message={
^pfmt^=10
^ver^=30
^type^=1
^encoding^=1
^tracking^={^id^="UX78vPDLVW/dR-lS9GzzwA0kzzw"}
^data^={
    SybCONTEXT={^class^="Context"
            repName="name 2"
            repCode="code 2"}
            SybNATIVEMSG=[588 opaque bytes]
            SybCOMMONMSG=[36 opaque bytes]
}
}
```

For an Active Enterprise wire-formatted message generator, the output is:

```
Data Received:
{, M_TREE {
    {^tracking^, M_TREE {
      {^id^, M_STRING, "UX78vPDLVW/dR-lS9GzzwA0kzzw"}
      }}
      {CONTEXT, M_TREE {
          {^class^, M_STRING, "Context"}
            {repName, M_STRING, "name 2"}
            {repCode, M_STRING, "code 2"}
      }}
      SybNATIVEMSGdbeventy?!<?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE dbStream SYSTEM
            'http://yjeongw2k:8000/repra/dtds/dbeventstream.dtd'>
      <dbStream environment="repraJMS2.repdb">
          <tran
              eventId="102:000000000000c8d500007fe7002a00007fe700280
00093bd00bd716e0000000000010001">
             <insert schema="REP4">
                 <values>
                    <cell name="repId" type="INT">2</cell>
                   <cell name="repName" type="VARCHAR">name 2</cell>
                   <cell name="repCode" type="VARCHAR">code 2</cell>
                 </values>
              </insert>
          </tran>
```

```
      </dbStream>
      }
      {SybCOMMONMSG, M_BINARY, $ +Ue^class^?SybCommonMSG_UDS }
      }}
```

Customizing the Sender and Formatter Processors

Customize the sender processor and the formatter processor for routing the incoming replication events to meet your application needs.

1. Develop your own Java class, implementing APIs that are provided by RepConnector.
2. Define the class in your connection configuration.
3. Modify the server environment.

Note: When you configure the RepConnector connection, you must indicate that you will be using a customized sender processor.

See also

- *Configuring RepConnector* on page 19

Customizing the Sender Processor

Use the API provided with RepConnector to create custom senders that can route replication events to anywhere that is accessible to Java, such as e-mail applications, files, or printers.

RepConnector has built-in senders that can post replication events to Sybase-certified JMS, TIBCO, and IBM MQ messaging systems. To create a customized sender processor that runs within the RepConnector environment:

1. Create a class that implements either `com.sybase.connector.repra.RepraClient`, or `com.sybase.connector.repra.RepraCustomClient`. `RepraCustomClient` allows the RepConnector Manager to set and load a property page, but `RepraClient` does not.
2. Compile the class and archive it to a `.jar` file.
3. Use RepConnector Manager to add a RepConnector connection that routes events to the custom sender processor:
   - Select REPLICATION for the inbound type, and CUSTOM for the outbound type.
   - On the Plug-in Class Information page of the wizard, enter the name of the custom class as the Sender Processor Plug-in Class. If `RepraCustomClient` is implemented, enter the path to and the name of the property page as Sender Processor Properties File.

4. Modify the application server environment to load the customized sender processor. See *Configuring an Application Server for a Custom Sender Processor or Formatter*.
5. Shut down and restart the application server.

See also

- *Configuring RepConnector* on page 19

RepraClient Interface

A RepraClient interface example.

```
package com.sybase.connector.repra;

public interface RepraClient
{
     /**
     *configures the sender properties and connects the sender/
receiver of the
     *target messaging system.
     */
     public void configureClient() throws Exception;

     /**
     *send out the rep messages to the connection.
     *@param String repmsg the text stream of the XML document
containing
     *the metadata and replication event.
     */
     public boolean sendEvent (Object repmsg) throws Exception;

     /**
     *return true if the connection is healthy.
     */
     public boolean isReady();

     /**
     *close the client connection.
     */
     public void close();

     /**
     *sets the default logger
     */
     public void setLogger (RaLogger log);
}
```

Sample Implementation for RepraClient Interface

```
package com.mycompany;

import com.sybase.connector.repra.RepraClient;
import com.sybase.connector.repra.logging.RaLogger;
```

```
import java.io.*;

public class SampleClient implements RepraClient
{
     BufferedWriter _fout;
     String _filename = "myCustomOut.dat"

     // This method creates an instance of the BufferedWriter object
     public void configureClient() throws Exception
     {
         _fout = new BufferedWriter(new FileWriter(_filename, true));
     }

     // You can do whatever you want in this method.
     // This sample appends the String value of the message to the
file.
     public boolean sendEvent(Object repmsg) throws Exception
     {
          _fout.write(repmsg.toString(), 0,
repmsg.toString().length());

          _fout.newLine();
          _fout.flush();

          return true;
     }

     //It returns true if the client channel is ready.
     //Otherwise, it returns false.
     public boolean isReady()
     {
          if (_fout != null)
          {
               return true;
          }
          return false;
     }

     // This method closes the client channel.
     public void close()
     {
          if (isReady())
          {
              try
              {
                   _fout.close();
              }
              catch (Exception ex)
              {
                   ex.printStackTrace();
              }
          }
     }

     // This method sets the default logger. In this sample, it
     // does nothing.
```

```
     public void setLogger(RaLogger log)
     {
     }
}
```

RepraCustomClient Interface

A RepraCustomClient Interface example.

```
package com.sybase.connector.repra;

/**
    Configures the custom sender and custom property page
*/
public interface RepraCustomClient extends RepraClient,
RepraCustomProps
{
}
```

Sample Implementation of the RepraCustomClient Interface

```
import java.io.FileInputStream;
import java.util.Date;
import java.util.Properties;import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;import
com.sybase.connector.repra.RepraCustomClient;
import com.sybase.connector.repra.logging.RaLogger;/*
 * SampleClient2
 *
 * Description:    This is a sample of a customer sender
 * processor client that will load a custom property
 * page and then route the event to an email address
 *
 */
public class SampleClient2 implements RepraCustomClient
{
    private Transport _transport = null;
    private MimeMessage _msg = null;
    private RaLogger _log = null;

    protected static String _host;

    protected String _from;

    protected String _to;

    protected String _cc;

    protected String _subject,

    protected String _username;
```

```
    protected String _password;    private String _propFile;    /**
    * sets the property file
    */    public void setConfigProps(String custPropsFile)
{
        _propFile = custPropsFile;
}
    /**
    * gets the property file
    */
    public String getConfigProps()
    {
    return _propFile;
    }
    /**
        * sets the default logger
        */
        public void setLogger (RaLogger log)
    {
    if (_log == null)
        {
            _log = log;
        }
    }
    /**
 * gets the email information from the properties
 * @throws Exception
        */
        private void getHostInformation() throws Exception
    {
        String thePropFile = getConfigProps();
        FileInputStream in = newFileInputStream(thePropFile);
        _log.info ("SampleClient2.INFO", "** IN
getHostInformation, loading prop file");
```

```
        Properties prop = new Properties();
        prop.load(in);
        _host = prop.getProperty("MAIL_HOST","");
        _from = prop.getProperty("MAIL_FROM","");
        _to = prop.getProperty("MAIL_TO","");
        _cc = prop.getProperty("MAIL_CC","");
        _subject = prop.getProperty("MAIL_SUBJECT","");
        _username = prop.getProperty("MAIL_USERNAME","");
        _password = prop.getProperty("MAIL_PASSWORD","");
        _log.info ("SampleClient2.INFO", "** HOST - " + _host + ",
MAIL_TO - " +_to);
        }
        public void configureClient() throws Exception
        {
        try
        {
            _log.info ("SampleClient2.INFO", "** Starting
ConfigureClient");
            Properties prop =System.getProperties();
            Session ses  =
Session.getDefaultInstance(prop,null);
            getHostInformation();
            _msg = new MimeMessage(ses);
            _msg.setFrom(new InternetAddress(_from));
            _msg.addRecipient(Message.RecipientType.TO, new
InternetAddress(_to));
            _msg.addRecipient(Message.RecipientType.CC, new
InternetAddress(_cc));
            _msg.setSubject(_subject);
            _msg.setSentDate(new Date());
            _msg.saveChanges();
            _transport = ses.getTransport("smtp");
            _transport.connect(_host, _username, _password);
```

```
            log.info ("SampleClient2.INFO", "** Ending
ConfigureClient");
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    /**
     * sends out the repmsg as an email
     */
    public boolean sendEvent(Object repmsg) throws Exception

        {
        try
        {
            _log.info ("SampleClient2.INFO", "** Starting
           SendEvent, repmsg is " + repmsg.toString());
          _msg.setText(repmsg.toString());
          _transport.sendMessage(_msg,
          _msg.getAllRecipients());
         return true;
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    /**
        * returns true if the connection is healthy
     */
    public boolean isReady()
    {
```

```
        return _transport.isConnected();
    }
        /**
     * closes the client connection
     */
     public void close()
        {
    try
        {
            _transport.close();
        }
        catch (Exception ex)
        {
            // do nothing
        }
    }
}
```

RepraCustomProps Interface

A RepraCustomProps Interface example.

```
package com.sybase.connector.repra;
/** configures getter and setter for custom property pages*/
public interface RepraCustomProps
{
    /**
    * Set the Customize Properties File
    *
    * @param custPropsFile path to the customize property file
        */
    public void setConfigProps(String custPropsFile);

    /**
    * Get the Customize Properties File
    *
    * @return The path to the customize property file
    */
    public String getConfigProps();
}
```

Customizing the Formatter Processor

Use the Jave API provided with RepConnector to create custom formatters that translate replication events into other formats. RepConnector translates these replication events into XML before delivering them.

1. Create a class that implements either `com.sybase.connector.repra.rep.RepTransactionFormatter` or `com.sybase.connector.repra.rep.RepraCustomTransactionFormatter`.

   `RepraCustomTransactionFormatter` allows the RepConnector Manager to set and load a property page, while `RepTransactionFormatter` does not.
2. Compile the class and archive it to a `.jar` file.
3. Use RepConnector Manager to add or modify a RepConnector connection where the inbound type is "REPLICATION."
   - On the General Information page, select **Customized Plug-in Class**.
   - On the Plug-in Class Information page of the wizard, enter the name of the custom class in the Message Formatter Plug-in Class field. If `RepraCustomTransactionFormatter` is implemented, enter the path to the property page and its name in the Message Formatter Properties File field.
4. Modify the application server environment to load the customized message formatter.
5. Shut down and restart the application server.

See also

• *Configuring an Application Server for a Custom Sender Processor or Formatter* on page 26

RepTransactionFormatter Interface

A RepTransactionFormatter Interface example.

```
package com.sybase.connector.repra.rep;

import com.sybase.connector.repra.logging.RaLogger

public interface RepTransactionFormatter
{
     /**
     returns an Object formatted by this formatter implementation.
     rse - the internal Object containing a replication event.
     */
     publicObject format(RepEvent rse) throws RepraException;

     /**
     returns an Object formatted by this formatter implementation.
```

```
     rse - The internal Object containing a replication event.
     */
     public Object formatTransaction(RepEvent[] events)
          throws RepraException;

     /**
     Return true if RepEvent metadata is required for formatting.
     If it returns true, the RepEvent will contain the data type of
     each field. Otherwise, the data type will be ignored for this
     RepEvent.
     */
     public boolean requiresMetaData();

     /**
    Return true if RepEvent is required to be parsed to be a standard
    RepEvent that the RepEventParser can handle. If it returns false,
    the RepEvent will contain a text stream of the RepEvent only for
the
     messaging client to parse it.
     */
     public boolean requiresParse();

     /**
     sets the default logger
     */
     public void setLogger(RaLogger log);
}
```

Sample Implementation of the RepTransactionFormatter Interface

Use the `MessageFormatter.java` file in the `<AppServer location>\repra\sample\client` directory on Windows, or `<AppServer location>/repra/sample/client` directory on UNIX, as a sample of the customized message formatter. The sample uses the **DBEventParser** utility to retrieve data and metadata from the replication event.

See also

- *DBEventParserFactory Utility* on page 75

Creating New Custom Sender and Custom Formatter Classes

Create a user-defined property page by implementing two interfaces: RepraCustomClient and RepraCustomTransactionFormatter.

Both interfaces add the same two methods to RepraClient and RepTransactionFormatter:

```
public void setConfigProps(String custPropsFile):
public String getConfigProps();
```

setConfigProps sets the user-defined property page, while **getConfigProps** gets the user-defined property page.

There are two samples, `CustomMessageFormatter.java`, and `MailClientCustom.java`, in the RepConnector installation's `sample/client` directory. MailClientCustom uses a custom configuration file named `sender.props`, which is also in the sample directory.

DBEventParserFactory Utility

RepConnector provides a utility called **DBEventParserFactory** that you can use to extract both metadata and actual data from a single or grouped replication event.

Use the utility with the customized formatter to extract and format data before sending it to the destination. It can also be used by an end-user application that has received the XML representation of the event.

To obtain a parser instance, use:

```
DBEventParser=dbe=DBEventParserFactory.get EventParser (xmldoc or
repevent)
```

DBEventParser APIs

Your customized formatter can use retrieved information to generate a new message in a customized format to send to the sender processor.

package com.sybase.connector.repra.util; setSource(Object obj) throws Exception

Sets the source event. Must be set prior to calling other methods that retrieve information. The `obj` that is passed to **setSource** is a `repevent[]` object or a string representation of the XML event.

Syntax

```
setDatabaseType (int dbType)
     DBEventParser.DBTYPE_ORACLE
                       DBType_SYBASE
```

int size()

Returns the number of operations in the transaction.

String getDSName() throws Exception

Returns the number of operations in the transaction.

String setDBName() throws Exception

Returns the database name defined for the connection.

String getEventId() throws Exception

Returns the unique event ID of this transaction.

String getOperation(int elemAt)

Returns the operation (**insert**, **delete**, **update**, or **exec**) at the position of **elemAt**. If there is only one element, use 0.

Example

```
int msgSize = size();
for (int ido = 0; ido < msgSize; ido++)
{
    // prints out the operation name of the (ido)th
    // operation
    System.out.println("MyMsgGenerator-Operation : " +
             dbe.getOperation(ido));
}
```

String getSchemaName(int elemAt) throws Exception

Returns the table name or the procedure name of the operation at the position of **elemAt**.

Example

```
int msgSize = size();
for (int ido = 0; ido < msgSize; ido++)
{
   // prints out the table name of the (ido)th operation
   System.out.println("MyMsgGenerator-TableName : " +
         dbe.getSchemaName(ido));}
```

String getStatement() throws Exception

Returns the SQL statement of the entire transaction. If the transaction contains only one operation, this API returns only one statement. If there are multiple operations in the transaction, this API returns multiple statements separated by the newline character (“\n”).

Example

```
int msgSize = size();
if (msgSize > 0)
{
System.out.println (dbe.getStatement());
}
```

Sample output:

```
insert into REP4 values (1, "code 1", "name 1")
insert into REP4 values (2, "code 2", "name 2")
update REP4 set repCode = "code 1111" where repId=1
```

String setStatement(int elemAt) throws Exception

Returns a single SQL statement belonging to the operation at the position of **elemAt**.

Example

```
int msgSize = size();
for (int ido = 0; ido < msgSize; ido++)
{
    // prints out the statement of the (ido)th operation
    System.out.println("MyMsgGenerator-Statement : " +
         dbe.getStatement(ido));
}
```

String getOwner(int elemAt) throws Exception

Returns the owner of the table or procedure used in this operation.

Example

```
dbe.getOwner(0);
```

Vector getData(int elemAt) throws Exception

Returns a vector containing hash tables which represent the names, types, and values of the fields belonging to the operation at the position of **elemAt**. This method is meaningful only for **insert**, **update**, and stored procedure operations.

The hash table includes this information:

```
DBEventParser.FIELD_NAME=(String) <field name>
DBEventParser.FIELD_TYPE=(Integer) <field type>
DBEventParser.FIELD_VALUE=(Object) <field value>
```

where:

```
DBEventParser.FIELD_NAME = "FieldName",
DBEventParser.FIELD_TYPE = "FieldType",
DBEventParser.FIELD_VALUE = "FieldValue".
```

Example

```
// Gets the fields of the first operation
Vector dataVector = dbe.getData(0);
Hashtable dataField = null;
if (dataVector != null)
{
   System.out.println("MyMsgGenerator-DataSize : " +
   dataVector.size());
   for (int ido = 0; ido < dataVector.size(); ido++)
   {
     // returns a Hashtable containg the name, type, and
     // value of the (ido)th field
     dataField = (Hashtable)      dataVector.elementAt(jdo);
```

```
     // Do something to retrieve the name, type and
     // value of the field

   }
}
```

Vector getKeys(int elemAt) throws Exception

Returns a vector containing hash tables that represent the names, types, and values of the key field belonging to the operation at the position of **elemAt**. This method is meaningful only for **update** and **delete** operations.

The hash table includes this information:

```
DBEventParser.FIELD_NAME=(String) <field name>
DBEventParser.FIELD_TYPE=(Integer) <field type>
DBEventParser.FIELD_VALUE=(Object) <field value>
```

where:

```
DBEventParser.FIELD_NAME = "FieldName",
DBEventParser.FIELD_TYPE = "FieldType",
DBEventParser.FIELD_VALUE = "FieldValue".
```

Example

```
// Gets the fields of the first operation
Vector keyVector = dbe.getKeys(0);
Hashtable keyField = null;
if (keyVector != null)
{
   System.out.println("MyMsgGenerator-KeySize : " +
   keyVector.size());
   for (int ido = 0; ido < keyVector.size(); ido++)
   {
     // returns a Hashtable containg the name, type,
     // and value of the (ido)th key field
     keyField = (Hashtable) keyVector.elementAt(jdo);
     // Do something to retrieve the name, type and
     // value of the key field
   }
}
```

String getFieldName(Hashtable field) throws Exception

Returns the field name of this column as a string value of `DBEventParser.`

int getFieldType(Hashtable field) throws Exception

Returns the field type of this column as an *int* value of DBEventParser.

JDBC Datatype Constant	Constant Value (int)
`DBEventParser.CHAR`	0

JDBC Datatype Constant	Constant Value (int)
DBEventParser.UNICHAR	25
DBEventParser.UNIVARCHAR	110
DBEventParser.BINARY	1
DBEventParser.TEXT	4
DBEventParser.UNITEXT	29
DBEventParser.IMAGE	5
DBEventParser.TINYINT	6
DBEventParser.SMALLINT	7
DBEventParser.INT	8
DBEventParser.BIGINT	30
DBEventParser.USMALLINT	31
DBEventParser.UINT	32
DBEventParser.UBIGINT	33
DBEventParser.REAL	9
DBEventParser.FLOAT	10
DBEventParser.BIT	11
DBEventParser.DATETIME	12
DBEventParser.SMALLDATETIME	13
DBEventParser.MONEY	14
DBEventParser.SMALLMONEY	15
DBEventParser.NUMERIC	16
DBEventParser.DECIMAL	17
DBEventParser.VARCHAR	18
DBEventParser.VARBINARY	19
DBEventParser.DATE	27

JDBC Datatype Constant	Constant Value (int)
DBEventParser.TIME	28

Object getFieldValue(Hashtable field) throws Exception

Returns the field value of this column as an object.

The subtype of the object is determined by these mappings:

Java Object	JDBC Datatype
Boolean	DBEventParser.BIT
ByteArrayInputStream	DBEventParser.BINARY DBEventParser.SMALLBINARY DBEventParser.IMAGE
Double	DBEventParser.REAL
Float	DBEventParser.FLOAT
Integer	DBEventParser.TINYINT DBEventParser.SMALLINT DBEventParser.INT DBEventParser.USMALLINT DBEventParser.UINT
java.math.BigDecimal	DBEventParser.UBIGINT
Long	DBEventParser.BIGINT

Java Object	JDBC Datatype
String	DBEventParser.CHAR DBEventParser.VARCHAR DBEventParser.UNICHAR DBEventParser.UNIVARCHAR DBEventParser.UNITEXT DBEventParser.DATETIME DBEventParser.SMALLDATETIME DBEventParser.MONEY DBEventParser.SMALLMONEY DBEventParser.NUMERIC DBEventParser.DECIMAL DBEventParser.TEXT DBEventParser.DATE DBEventParser.TIME

Example

This example shows **getFieldName**, **getFieldType**, and **getFieldValue**.

```
int msgSize = size();
// This iteration visits all of the operations
for (int ido = 0; ido < msgSize; ido++)
{
   System.out.println("MyMsgGenerator-Statement:" +
     getStatement(ido));
   System.out.println("MyMsgGenerator-Operation:" +
     getOperation(ido));
   System.out.println("MyMsgGenerator-TableName:" +
     getSchemaName(ido));
   Vector dataVector = getData(ido);
   Hashtable dataField = null;
   if (dataVector != null)
   {
     System.out.println("MyMsgGenerator-DataSize: " +
     dataVector.size());
     // This iteration visits the fields of the
     // operation
     for (int jdo = 0; jdo < dataVector.size(); jdo++)
     {
       dataField = (Hashtable) dataVector.elementAt(jdo);
       if (dataField == null)
```

```
       {
         break;
       }
       System.out.println("MyMsgGenerator-FieldName:"+
         getFieldName(dataField));
       System.out.println("MyMsgGenerator-FieldType:"+
         getFieldType(dataField));
       System.out.println("MyMsgGenerator-FieldValue:" +
         getFieldValue(dataField).toString);
         }
       }
     }
```

String toXMLText(String dtdURL) throws Exception

Returns an XML text-type (a string) containing the events of the transaction.

Example

```
System.out.println(toXMLText
    (http://yjeongw2k:8000/RepraWebApp/dtds
    /dbeventStream.xsd));
```

Sample output:

```
   <!DOCTYPE dbStream SYSTEM
     'http://yjeongw2k:8000/RepraWebApp/dtds/dbeventstream.xsd'>
   <dbStream environment="repraJMS2.repdb">
     <tran eventId=
"102:0000000000000ab200003e10004b00003e1000490000937300dda250000000
00000
10001">
       <update schema="REP4">
         <values>
           <cell name="repId" type="INT">2</cell>
           <cell name="repName" type="VARCHAR">name 11</cell>
           <cell name="repCode" type="VARCHAR">code 11</cell>
         </values>
         <oldValues>
           <cell name="repId" type="INT">11</cell>
         </oldValues>
       </update>
     </tran>
   </dbStream>
```

RaXMLBuilder Utility

The **RaXMLBuilder** utility generates an XML message format that contains the database events that a user application sends to RepConnector, to be routed to the database.

RaXMLBuilder()

The default constructor.

Syntax

```
Package: com.sybase.connector.repra.utility
Constructor RaXMLBuilder()
```

createTranDocument() throws Exception

Creates a document with the <tran> element, to contain multiple database operations in a transaction. Returns a String, the Element of the current event.

Syntax

```
org.dom4j.Element createTranDocument
      (java.lang.String uri,java.lang.String dbname,
       java.lang.String eventId)
```

Parameters

- *uri* – the URI of dbevenstream.xsd.
- *dbname* – the database on which the operation executes.
- *eventId* – the event ID of the current transaction. The uniqueness of eventId is the responsibility of the sending client.

createEventDocument() throws Exception

Creates a document with the element to contain a single database operation. Returns a String, the Element of the current event.

Syntax

```
org.dom4j.Element createEventDocument
         (java.lang.String uri, java.lang.String dbname,
           java.lang.String eventId)
```

Parameters

- *uri* – the URI of dbeventstream.xsd.
- *dbname* – the database on which the operation is executed.
- *eventId* – the event ID of the current transaction. The uniqueness of eventID is the responsibility of the sending client.

addOperation() throws Exception

Adds the database operation to the current event, either the <tran> element or the <dbEvent> element. If the event type exists and already contains an operation,

addOperation()throws Exception returns null. Otherwise, it returns a `String`, the Element of the current operation.

Syntax

```
org.dom4j.Element addOperation(java.lang.String operName,
java.lang.String schemaName)
```

Parameters

- *operName* – the SQL operation, such as **insert**, **update**, **delete**, or **exec**.
- *schemaName* – the target table.

addValue() throws Exception

Adds field data to the operation.

Syntax

```
void addValue (org.dom4j.Element operElem,java.lang.String
fieldName,
 java.lang.String fieldType, java.lang.String fieldValue)
```

Parameters

- *operElem* – `Element`.
- *fieldName* – `String`.
- *fieldType* – `String`. The JDBC-SQL datatype.
- *fieldValue* – `String`. The entire value must be passed as `String`.

addInValue() throws Exception

Adds the data of the input field to the operation for a stored procedure.

Syntax

```
void addInValue(org.dom4j.Elem operElem,
                    java.lang.String fieldName,
                    java.lang.String fieldType,
              java.lang.String fieldValue)
```

addOutValue() throws Exception

Adds the data of the output field to the operation for a stored procedure.

Syntax

```
void addOutValue(org.dom4j.Elem operElem,
                    java.lang.String fieldName,
                    java.lang.String fieldType,
               java.lang.String fieldValue)
```

addWhere() throws Exception

Adds a **where** clause to the operation, using **AND** as the default condition and **=** as the default operator.

Syntax

```
void addWhere(org.dom4j.Elem operElem,
                    java.lang.String fieldName,
                    java.lang.String fieldType,
                java.lang.String fieldValue)
```

```
void addWhere(org.dom4j.Elem operElem,
                    java.lang.String fieldName,
                    java.lang.String fieldType,
                    java.lang.String fieldValue,
                    java.lang.String condition,
              java.lang.String operator)
```

Parameters

- *condition* – **AND** or **OR**.
- *operator* – a SQL operator.

write() throws Exception

Prints XML text to the specified file.

Syntax

```
void write(java.lang.String filename)
```

Parameters
filename – the target file.

xmlDocByteArray() throws Exception

Returns a **ByteArrayOutputStream** containing the XML data.

Syntax

```
java.io.ByteArrayOutputStream xmlDocByteArray()
```

xmlDocString() throws Exception

Returns a `String` containing XML data.

Syntax

```
java.lang.String xmlDocString()
```

cancelOperation() throws Exception

Drops the operation element from the root event.

Syntax

```
void cancelOperation(org.dom4j.Element elem)
```

Parameters

elem – the operation element to be cancelled.

getErrorEventId() throws Exception

Returns the event ID of the message that caused the error message.

Syntax

```
static java.lang.String getErrorEventId(java.lang.String xmlText)
```

Parameters

xmlText – the `String` value of the error document.

getErrorStatusCode() throws Exception

Returns the error code from the error document.

Syntax

```
static java.lang.String getErrorStatusCode(java.lang.String xmlText)
```

getErrorMessage() throws Exception

Returns the error message from the error document.

Syntax

```
static java.lang.String getErrorMessage(java.lang.String xmlText)
```

String getOwner(int elementAt) throws Exception

Retrieves the owner name of the replication event.

Example

```
           System.out.println("Owner of the
table:"+_parser.getOwner(0));
```

Configuring the RaXMLBuilder

To configure the RaXMLBuilder, include the `repraconn.jar` file in your CLASSPATH environment variable setting.

For Windows, enter:

```
SET CLASSPATH=%REPRA_HOME%\lib\repraconn.jar;%CLASSPATH%
```

For UNIX, enter:

- For **bsh**:
  ```
  CLASSPATH =$REPRA_HOME/lib/repraconn.jar:$CLASSPATH export
  CLASSPATH
  ```
- For **csh**:
  ```
  setenv CLASSPATH $REPRA_HOME/lib/repraconn.jar:$CLASSPATH
  ```

Using the RaXML Utility

Use the **RaXML** utility in your code.

1. Import the essential modules:
   ```
   import org.dom4j.Element;
   import com.sybase.connector.repra.util.*;
   ```
2. Create an instance of RaXMLBuilder:
   ```
   RaXMLBuilder raXML = new RaXMLBuilder();
   ```
3. Get the event body, which requires three parameters:
   - The URI of the `xsd.` file
   - The name of the database
   - The event ID of the current event, which can be any string value

     If you want the transaction (<tran>) type to contain multiple database operations, enter:
     ```
     foo.createTranDocument("file://dbeventstream.
                 xsd","pubs2"","00001001");
     ```

     If you want the event (<dbevent>) type to contain a single database operation, enter:
     ```
     foo.createEventDocument("file://dbeventstream.
                 xsd","pubs2"","00001001");
     ```
4. Add an operation, which requires:
   - The command
   - The name of the schema

   For example:

```
Element oper1=foo.addOperation("update","authors"):
```

5. Add data to the operation, which requires:

   - The operation *element*
   - *fieldName*
   - *fieldType*
   - *fieldValue*, the string value of the field

   For example:

   ```
   foo.addValue(operl,"au_id","CHAR","0001");
   foo.addValue(oper1,"au_num","INT","1");
   ```

   The field types, as SQL datatypes, are:

   - `TEXT`, `DATETIME`, `SMALLDATETIME`, `MONEY`, `SMALLMONEY`,
   - `NUMERIC`, `DECIMAL`, `VARCHAR`, `CHAR`, `DATE`, `TIME`
   - `BINARY`, `IMAGE`, `VARBINARY`
   - `TINYINT`, `SMALLINT`, `INT`
   - `REAL`
   - `FLOAT`
   - `BIT`
   - `UNICHAR`, `UNIVARCHAR`
   - `UNITEXT`
   - `BIGINT`, `USMALLINT`, `UINT`, `UBIGINT`

6. Add a **where** clause to the operation, which requires:

   - The operation element
   - *fieldName*
   - *fieldType*
   - *fieldValue*
   - SQL condition: either **AND** or **OR**
   - SQL operator: **=**, <,>, **NOT**, and so forth

   For example:

   ```
   foo.addWhere
           (oper1,"au_id","CHAR","0002","AND","=");
   ```

7. Create an XML file:

   ```
   foo.write(fileName);
   ```

8. Get the `String` value of the event from the current XML document:

   ```
   String dataStr=foo.xmlDocString();
   ```

   Your application must send the `dataStr` object to the RepConnector connection.

Running a Sample Implementation

Compile and run the sample, and include the `repraconn.jar` file in your CLASSPATH.

Prerequisites

Include the `repraconn.jar` file in your CLASSPATH.

Task

A sample implementation, `UseXMLBuilder.java`, is in the RepConnector installation `sample/client` directory. To run the sample:

- On Windows, enter:

```
java- classpath.:%REPRA_HOME%\lib\repraconn.jar:
%REPRA_HOME%    \lib\dom4j-full.jar
UseXMLBuildertext.xml
```

- On UNIX, enter:

```
java- classpath.:$REPRA_HOME/lib/repraconn.jar:$REPRA_
HOME/lib/dom4j-full,jar UseXMLBuildertest.xml
```

The result:

```
Output of multiple update db events in a single transaction:
         <?xml version="1.0" encoding="UTF-8"?>
```

```
  dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file://dbeventstream.xsd"
   environment="pubs2">
   <tran eventId="00001001">
```

```
   <update schema="authors">
    <values>
     <cell name="au_id" type="CHAR">0001</cell>
     <cell name="address" type="VARCHAR">1 Sybase</cell>
   <values>
   <oldValues>
    <cell name="au_id" type="CHAR" operator="=">0002</cell>
    <cell name="address" type="VARCHAR" condition="AND"
    operator="=">3 Sybase</cell>
   </oldValues>
  </update>
    <delete schema="authors">
     <values>
       <cell name="au_id" type="CHAR">0001</cell>
       <cell name="address" type="VARCHAR">1 Sybase</cell>
     </values>
   </oldValues>
     <cell name="au_id" type="CHAR" operator="=">0002</cell>
     <cell name="address" type="VARCHAR" condition="AND"
     operator="=">3 Sybase</cell>
     </oldValues>
    </delete>
    </update>
```

```
   </tran>
  </dbStream>

Output of multiple dbevents in a transaction:
<?xml version="1.0" encoding="UTF-8"?>

   dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file://dbeventstream.xsd"
   environment="pubs3">
 <tran eventId="00001002">
   <update schema="stores">
    <values>
     <cell name="au_id" type="CHAR">0002</cell>
     <cell name="address" type="VARCHAR">1 Sybase</cell>
   <values>
  </oldValues>
 </update>
 <exec schema="storesProcedure1">
   <inValues>
     <cell name="au_id" type="CHAR">0002</cell>
   </inValues>
   <outValues>
     <cell name="au_id" type="CHAR">0002</cell>
   </outValues
  </exec>
 </dbEvent>
</dbStream>


Output of a stored procedure:
<?xml version="1.0" encoding="UTF-8"?>
   dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file://dbeventstream.xsd"
   environment="pubs3">
 <dbEventId="00001003">
   <update schema="storesProcedure1">
<inValues>
     <cell name="au_id" type="CHAR">0002</cell>
   </inValues>
   <outValues>
     <cell name="au_id" type="CHAR">0002</cell>
   </outValues
  </exec>
 </dbEvent>
</dbStream>
```

Handling Error Messages

RepConnector sends any errors it encounters while processing an event to your Configure Status message queue. When you receive an error message from the queue, parse the error message for the `eventId`, `errorCode`, and the message itself.

For example:

```
System.out.println("Error EventId:"+
   RaXMLBuilder.getErrorEventId(err));
System.out.println("Error StatusCode:"+
   RaXMLBuilder.getErrorStatusCode(err));
System.out.println("Error Message:"+
      RaXMLBuilder.getErrorMessage(err));
```

Compiling and Running the Sample

To build and send XML data to the TIBJMS queue, use the files for compiling and running the sample JMS client.

- On Windows, enter:

```
%REPRA_HOME%\sample\client\tibjms
%REPRA_HOME%\sample\client\tibjmssetup.bat
%REPRA_HOME%\sample\client\runTIBJMSQSender.bat
```

- On UNIX, enter:

```
$REPRA_HOME/sample/client/tibjmsClientSender.java
$REPRA_HOME/sample/client/tibjmssetup.sh
$REPRA_HOME/sample/client/runTIBJMSQSender.bat
```

Managing Ownership Information

Use the XML utility schema to manage ownership information about tables that have the same name, but different owners.

To use a copy of the `dbeventstream.dtd` or `dbeventstream.xsd` to parse XML data generated by RepConnector after you install this API, you must upgrade the file in `%REPRA_HOME%\dtds`.

For example:

```
<your application server installation directory>\repra directory>
```

If you configure a replication definition for ownership of the table, RepConnector includes the owner name of the table or stored procedure in the output XML data.

Output XML data without ownership information:

```
        <?xml version="1.0" encoding="UTF-8"?>
        <dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance
     xsi:noNamespaceSchemaLocation="http://localhost:8000/
RepraWebApp/dtds
        /dbeventstream.xsd" environment="RepConn.repdb">
        <tran
eventId="102:0000000000028cfc000007d8000f000007d8000c0000955700c078
9e000
          000000001001">
        <delete schema=RepTable3">
         <oldValues>
          <cell name="repId" type="INT">1</cell>
         </oldValues>
        </delete>
```

```
        </tran>
      </dbStream>
```

Output XML data with ownership information:

```
        <?xml version="1.0" encoding="UTF-8"?>
        <dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance
     xsi:noNamespaceSchemaLocation="http://localhost:8000/
RepraWebApp/dtds
        /dbeventstream.xsd" environment="RepConn.repdb" owner="dbo">
        <tran
eventId="102:0000000000028cfc000007d8000f000007d8000c0000955700c078
9e000
        000000001001">
        <delete schema=RepTable3">
         <oldValues>
          <cell name="repId" type="INT">1</cell>
         </oldValues>
        </delete>
        </tran>
      </dbStream>
```

Configuration Worksheets

You can use the worksheets to record all the configuration information for your RepConnector environment, Replication Server information, database information, application server information, messaging system information, and RepConnector information.

Fill out the worksheets as you configure each component in the system. Make a copy for each RepConnector connection profile and/or messaging system you have.

Table 3. Replication Server Information

	Current Value	Example	Maps to RepConnector Manager Wizard
DSI info			
1) Used by Replication Server to identify where RepConnector connection will run; this is the information added to the Replication Server `interfaces` file. 2) Used when the **create connection** command is executed at Replication Server to add the connection, including setting the user name and password. 3) Used in subscription for RepConnector at Replication Server.			
DSI Name	3.a	RepConnector	Replication Server Inbound Information page of New Connection wizard
Protocol	3.b	TCP	
DSI Host Name	3.c	localhost	
DSI Port	3.d	7000	
DSI User Name	3.e	sa	
DSI Password	3.f		
Replication Server System Database Information			
1) Used by Replication Server. 2) Used by RepConnector.			
RSSD Name	3.g	RepServer_RSSD	Replication Server System Database Information window of New Connection wizard.
RSSD Host Name	3.h	localhost	
RSSD Port Number	3.i	5000	
Replicated Database Information Needed by Replication Server			

	Current Value	Example	Maps to RepConnector Manager Wizard
1) Used in **create table** for replication at database. 2) Used in **create replication definition** for RepConnector at Replication Server.			
Primary DB to be Replicated	3.j	pubs2	Not needed in the New Connection wizard
Host Name of Database Server	3.k	primary_ase	
Port Number where DB is Listening	3.l	5000	
User Name	3.m	sa	
Password	3.n		

Table 4. JMS System Information

	Current Value	Example	Maps to RepConnector Manager Wizard
Destination Type		queue	Outbound or inbound messaging – JMS information
JMS Provider URL		jnp://localhost:1099	
Initial Naming Context Factory		org.jnp.interfaces.NamingContextFactory	
Connection Factory		java:/ConnectionFactory	
Destination Name		queue/sampleQueue	
User Name		guest	
Password			
Topic Subscribers			
Status Destination			

Table 5. TIBCO RBV Messaging System Information

	Current Value	Example	Maps to RepConnector Manager wizard
Message Type		RCVM	Outbound or inbound messaging – TIBCO messaging information
MQ Daemon		7500	
Encoding Type		localhost	
Host Name		tcp\:7500	
Channel		sample.Subject	
Queue Manager/Factory			
Queue Name		SAMPLE.CM1	
User Name		0	
Password			
Status Definition			

Table 6. IBM WebSphereMQ Messaging System Information

	Current Value	Example	Maps to RepConnector Manager wizard
Message Type		MQ	Outbound or inbound messaging – MQ messaging information
MQ Daemon			
Encoding Type		utf	
Host Name		localhost	
Channel		Channel1	
Queue Manager/Factory		MyManager	
Queue Name		SAMPLEQ	
Username		mquser	
Password		mqpass	
Status Definition			

Table 7. Database System Information

	Current Value	Example	Maps To
JDBC Connection URL		jdbc:sybase:Tds:local-host: 5000	Outbound type properties for an outbound type of database
Driver Class		com.sybase.jdbc2.jdbc.SybDriver	
User Name		sa	
Password			

Table 8. Customization Information

	Current Value	Example	Maps To
Message Formatter Plug-in Class		sample.MyMessageFormatter	Customization
Message Formatter Properties		sample.MyMessageFormatter.prop	
Sender Processor Plug-in Class		sample.FileClient	
Sender Processor Properties		sample.FileClient.prop	

Table 9. RepConnector Profile Properties

	Current Value	Example	Maps To
Profile Name		JBoss:8080	RepConnector Manager, Connection Profiles
Host Name		localhost	
Port Number		8080	
User Name		repraadmin	
Password			

Table 10. RepConnector Connection: General Properties

	Current Value	Example	Maps To
Inbound Type		Replication	RepConnector Manager, General Properties
Outbound Type		JMS	

	Current Value	Example	Maps To
DBEventStream XSD URL		http://localhost:8080/RepraWebApp/dtds/dbeventstream.xsd	
Log Level		INFO	
Auto Start Connection		FALSE	
Customized Plug-in Class		FALSE	

Table 11. Inbound RepConnector Connection: DSI Properties

	Current Value	Example	Maps To
DSI Name		RepConnector	RepConnector Manager, Inbound Properties
DSI Port		7000	
User Name		sa	
Password			

Table 12. Inbound RepConnector Connection: RSSD Properties

	Current Value	Example	Maps To
RSSD URL		jdbc:sybase:Tds:localhost:5000/REP_RSSD	RepConnector Manager, Inbound Properties
User Name		sa	
Password			
Grouping		FALSE	

Troubleshooting

Troubleshoot login, connection, and replication system problems.

Profile Login or ratool Failure

If you cannot log in to the RepConnector connection profile, check the application server environment, machine name and port number, and user name and password. You can also configure the RepConnector for debugging.

Verifying Application Server Environment

Verify that the application server is running, and has successfully called `repra_env.bat` on Windows, or `repra_env.sh` on UNIX.

1. Add an echo statement to `repra_env.bat` or `repra_env.sh`.
   - On Windows:
     ```
     echo SERVER_TYPE : %SERVER_TYPE%
     ```
   - On UNIX:
     ```
     echo SERVER_TYPE : $SERVER_TYPE
     ```
2. For WebLogic servers, verify that `repra_env.bat` or `repra_env.sh` is called correctly.

   `repra_env` must be called from the WebLogic startWebLogic command (**startWebLogic.cmd** on Windows, **startWebLogic.sh** on UNIX), in the `domain/bin` directory.

UNIX Example

```
/path/to/WebLogic_Domain/mydomain/bin/startWebLogic.sh
```

:

Before adding the **repra_env** call, **startWebLogic.sh** looks like this:

```
# Call setDomainEnv here
DOMAIN_HOME="/path/to/WebLogic_Domain/mydomain"
```

```
.${DOMAIN_HOME}/bin/setDomainEnv.sh$*
SAVE_JAVA)OPTIONS="${JAVA_OPTIONS}"
SAVE_CLASSPATH="${CLASSPATH}"
```

After adding **repra.env** call, you see **startWebLogic.sh** like this:

```
# Call setDomainEnv here.
DOMAIN_HOME="/path/to/WebLogic_Domain/mydomain"
```

```
.${DOMAIN_HOME}/bin/setDomainEnv.sh$*
if [-f/WebLogic_installation_directory/repra/bin/repra_env.sh]
then
./WebLogic_installation_directory/repra/bin/repra_env.sh
fi

SAVE_JAVA_OPTIONS="${JAVA_OPTIONS}"
SAVE_CLASSPATH="${CLASSPATH}"
```

Verifying Machine Name and Port Number

Use the log files to troubleshoot or verify the machine name and port number for your application servers.

For JBoss:

`<jboss_home_dir>\server\<server_name>\log\server.log`

For WebLogic Server:

- On Windows – `%WL_HOME%\user_projects\domains\`*`DomainName`*`\servers\`*`DomainNameServer`*`\logs\`*`DomainNameServer`*`.log`, where `%WL_HOME%` points to the application server installation directory.
- On UNIX – `$WL_HOME/user_projects/domains/`*`DomainName`*`/servers/`*`DomainNameServer`*`/logs/`*`DomainNameServer`*`.log`, where `$WL_HOME` points to the application server installation directory.

Verifying User Name and Password

If you change the default, run `setlogin.bat` on Windows or `setlogin.sh` on UNIX, before you attempt to log in to RepConnector Manager. The default user name for RepConnector Manager is “repraadmin” with a blank password.

See also

- *Get Started with RepConnector Manager* on page 15

Setting the Logging Level to DEBUG for the RepConnector Runtime Component

Configure RepConnector to collect logging information you can use for debugging runtime component.

1. From the command prompt, navigate to:
   - On Windows:
     ```
     cd <AppServer_install_dir>\repra\bin
     ```
   - On UNIX:
     ```
     cd <AppServer_install_dir>/repra/bin
     ```

2. In the `repra.properties` file, change the log level and runtime log level to DEBUG.

```
LogLevel=DEBUG
```

```
RuntimeLogLevel=DEBUG
```

3. Shut down and restart the application server.

Setting the Logging Level to DEBUG for Each RepConnector Connection

Configure RepConnector to collect logging information you can use for debugging connection.

1. Use RepConnector Manager to log in to RepConnector runtime component.
2. Modify the connection properties to change the logging level (LogLevel) for the connection to DEBUG. Save the new connection properties.
3. Start or refresh the connection.

Note: Setting the log level to DEBUG creates many debugging messages in the `repra.log` file. Use this information to troubleshoot failures, but also be aware that it may cause performance degradation.

Connection Failure

When a connection fails, look at the logs to troubleshoot connection and validation errors. To ensure that the log captures events, turn on **debug** mode.

This is a sample from a log in detail mode.

```
[RS_to_JMS]: 2013-01-15 02:07:53,243 [INFO] [REP.RepAdapterImpl]:
Starting Connection RS_to_JMS.
[RS_to_JMS]: 2013-01-15 02:07:53,245 [INFO] [JMS.JMSQueueClient]:
Starting the JMS Queue Client.

[RS_to_JMS]: 2013-01-15 02:07:53,250 [INFO] [JMS.JMSQueueClient]:
JMS Client is configured successfully to be able to send event(s) to
queue:
Queue1 for provider jnp://localhost:1099.

[RS_to_JMS]: 2013-01-15 02:07:53,250 [DEBUG] [REP.RepAdapterImpl]:
Successfully established client connection.

[RS_to_JMS]: 2013-01-15 02:07:53,252 [DEBUG]
[REP.RepMetaConnection]:
Executing query: >>>> select dbname from rs_databases where dsname =
'TEST_DSI_NAME'

[RS_to_JMS]: 2013-01-15 02:07:53,252 [DEBUG]
[REP.RepMetaConnection]: Got the new instance of SybDriver

[RS_to_JMS]: 2013-01-15 02:07:53,470 [DEBUG]
```

```
[REP.RepMetaConnection]: Got a RSSD connection to URL:
jdbc:sybase:Tds:replinuxb20:9545/REP_RS_ERSSD?SQLINITSTR=set
quoted_identifier off Login: REP_RSSD_prim

[RS_to_JMS]: 2013-01-15 02:07:53,472 [DEBUG]
[REP.RepMetaConnection]: Executing query: >>>>
select distinct rs_objects.objname, rs_subscriptions.subid,
rs_objects.objid, rs_objects.phys_tablename,
rs_objects.deliver_as_name,
rs_objects.dbid, rs_objects.phys_objowner from rs_repdbs,
rs_subscriptions, rs_objects where
((dsname = 'TEST_DSI_NAME' and dbname = 'fakename') and
rs_repdbs.dbid = rs_subscriptions.dbid and rs_subscriptions.type
< 8 and rs_subscriptions.objid = rs_objects.objid)  union select
distinct rs_objects.objname, rs_subscriptions.subid,
rs_objects.objid, rs_objects.phys_tablename,
rs_objects.deliver_as_name, rs_objects.dbid,
rs_objects.phys_objowner from rs_repdbs,
rs_articles, rs_subscriptions, rs_objects where  ((dsname =
'TEST_DSI_NAME' and dbname = 'fakename') and rs_repdbs.dbid =
rs_subscriptions.dbid
and rs_subscriptions.type > 8 and rs_articles.articleid =
rs_subscriptions.objid and rs_objects.objid = rs_articles.objid)

[RS_to_JMS]: 2013-01-15 02:07:53,479 [DEBUG]
[REP.RepMetaConnection]: Executing query: >>>> select * from
rs_columns
where objid = 0x0100006500000065 order by colnum

[RS_to_JMS]: 2013-01-15 02:07:53,480 [DEBUG] [REP.DataServer]: A new
RepListener was added to the RepEventStream.

[RS_to_JMS]: 2013-01-15 02:07:53,480 [DEBUG] [REP.RepEventStream]:
Added a Listener without RequiredGroup option

[RS_to_JMS]: 2013-01-15 02:07:53,480 [DEBUG] [REP.RepEventStream]:
RepAdapterListener requires parsing and meta-data formatting.

[RS_to_JMS]: 2013-01-15 02:07:53,480 [DEBUG] [REP.DataServer]: A new
listener is added to the stream object.

[RS_to_JMS]: 2013-01-15 02:07:53,481 [DEBUG] [REP.ReplicateDB]:
Created the dsi (.ser) file : /replinuxb20-1/users/repconnector/
jboss-as-7.1.1.Final/repra/sers/DSI_TEST_DSI_NAME_fakename.ser

[RS_to_JMS]: 2013-01-15 02:07:53,481 [DEBUG]
[REP.RepMetaConnection]: Executing query: >>>> select dsname,
dbname from rs_databases where dsname = 'TEST_DSI_NAME' and dbname =
'fakename'

[RS_to_JMS]: 2013-01-15 02:07:53,482 [DEBUG] [REP.DataServer]: Put
the DB connection to the Hashtable of ReplicationDBs.

[RS_to_JMS]: 2013-01-15 02:07:53,482 [DEBUG]
[REP.RepMetaConnection]: Executing query: >>>> select distinct
rs_objects.objname, rs_subscriptions.subid, rs_objects.objid,
```

```
rs_objects.phys_tablename,  rs_objects.deliver_as_name,
rs_objects.dbid, rs_objects.phys_objowner from rs_repdbs,
rs_subscriptions, rs_objects where
((dsname = 'TEST_DSI_NAME' and dbname = 'TEST_DB') and rs_repdbs.dbid
= rs_subscriptions.dbid and rs_subscriptions.type
< 8 and rs_subscriptions.objid = rs_objects.objid)  union select
distinct rs_objects.objname, rs_subscriptions.subid,
rs_objects.objid,
 rs_objects.phys_tablename,  rs_objects.deliver_as_name,
rs_objects.dbid, rs_objects.phys_objowner from rs_repdbs,
rs_articles,
rs_subscriptions, rs_objects where  ((dsname = 'TEST_DSI_NAME' and
dbname = 'TEST_DB') and rs_repdbs.dbid = rs_subscriptions.dbid
and rs_subscriptions.type > 8 and rs_articles.articleid =
rs_subscriptions.objid and rs_objects.objid = rs_articles.objid)

[RS_to_JMS]: 2013-01-15 02:07:53,488 [DEBUG]
[REP.RepMetaConnection]: Executing query: >>>> select * from
rs_columns
where objid = 0x0100006500000066 order by colnum[RS_to_JMS]:
2013-01-15 02:07:53,491 [DEBUG] [REP.RepMetaConnection]:
Executing query: >>>> select * from rs_columns where objid =
0x0100006500000067 order by colnum

[RS_to_JMS]: 2013-01-15 02:07:53,492 [DEBUG] [REP.DataServer]: A new
RepListener was added to the RepEventStream.

[RS_to_JMS]: 2013-01-15 02:07:53,492 [DEBUG] [REP.RepEventStream]:
Added a Listener without RequiredGroup option

[RS_to_JMS]: 2013-01-15 02:07:53,492 [DEBUG] [REP.RepEventStream]:
RepAdapterListener requires parsing and meta-data formatting.

[RS_to_JMS]: 2013-01-15 02:07:53,492 [DEBUG] [REP.DataServer]: A new
listener is added to the stream object.

[RS_to_JMS]: 2013-01-15 02:07:53,492 [DEBUG] [REP.ReplicateDB]:
Created the dsi (.ser) file :
/replinuxb20-1/users/repconnector/jboss-as-7.1.1.Final/repra/sers/
DSI_TEST_D
```

See also

- *Setting the Logging Level to DEBUG for the RepConnector Runtime Component* on page 100
- *Setting the Logging Level to DEBUG for Each RepConnector Connection* on page 101

Verifying Connection Information

Set the URL of the schema file, `dbeventstream.xsd`, to the application server's HTTP host name and port number of the application server, at the location of `RepraWebApp`.

For example:

```
http://localhost:8000/RepraWebApp/dtds/dbeventstream.xsd
```

This URL is placed in the XML message when it is generated, and states where the `xsd` can be found.

Verifying Replication Server Inbound Information

Verify that inbound information fields are set correctly.

- DSI Name – must be exactly the same name as added to Replication Server `sql.ini` file on Windows or to the `interfaces` file on UNIX, and the connection name used when creating the connection in Replication Server.
- Port – can be any available port; however, should be the same port identified in the `sql.ini` file on Windows or the `interfaces` file on UNIX.
- User Name or password – the same name identified when you created the connection in Replication Server.

See also

- *Configuration Worksheets* on page 93

Verifying Replication Server System Database Information

Verify Replication Server system database information.
Check for:

- RSSD URL, which should be in this format: `jdbc:Sybase:Tds:<ServerName>:port` for the location of the Replication Server
- User Name and password used in Replication Server for connectivity
  Use the **ping** feature in RepConnector Manager to validate user name and password.

Inbound or Outbound Messaging Systems Verification

Common verification error messages and a solution for each.

Problem

`Repra.log` example entry: java.lang.ClassNotFoundException.

Workaround

Check the RepConnector environment file:

- On Windows: <AppServer_install_dir>\repra\bin\repra_env.bat
- On UNIX: `<AppServer_install_dir>/repra/bin/repra_env.sh`

Note: On UNIX, directory names are case-sensitive.

Problem

Failed to get the JMS queue: test.sample for provider.

Workaround

Verify that the JMS Server is running and that the queue or topic has been correctly identified.

Problem
ping fails for IBM MQ when all information is correct.

Workaround
Stop any queue managers that are running and restart the queue manager you are identifying.

Use the **ping** feature in the RepConnector Manager to validate the status of inbound or outbound messaging systems.

Verifying Database Connection Information

Verify database connection information.

1. Check for:
   - JDBC connection URL, which should be in this format: `jdbc:Sybase:Tds:<ServerName>:port`
   - User name and password
2. Use the **ping** feature in RepConnector Manager to validate the status of the database connection information.

Troubleshooting the Replication System

Use the troubleshooting techniques for Adaptive Server Enterprise and Replication Server to fix the replication system problems.

Troubleshooting Adaptive Server Enterprise (Primary Database)

Use `error.log` to display a detailed problem description.

1. Access the `error.log` file:
   - On Windows, the error log file is `%SYBASE_ASE%\install\error.log`, where `%SYBASE_ASE` points to the Adaptive Server installation directory.
   - On UNIX, the error log file is `$SYBASE_ASE/install/error.log`, where `$SYBASE_ASE` points to the Adaptive Server installation directory.
2. To investigate the configuration of the primary database, use **isql** to log in to the primary database:

   ```
   isql -S <Server_name> -U <username> -P <password>
   ```

   For example, enter:

   ```
   isql -S primary_db -U sa -P sa_pass
   ```

Adaptive Server Commands

Use Adaptive Server commands to troubleshoot the primary database.

Command	Action
sp_start_rep_agent <dbname>	Start RepAgent
sp_stop_rep_agent <dbname>	Stop RepAgent
sp_setreplicate <tablename>, "true"	Set table for replication
sp_setreplicate	Give status of replicated tables
sp_setrepproc <proc_name>,function	Set the specified procedure for replication

Troubleshooting Replication Server

Use the Replication Server log file for a detailed problem description.

1. Access the log file:
   - On Windows, the log file is `%SYBASE%\REP-15_2\install\<RepServerName>.log`, where `%SYBASE_REP%` points to the Replication Server installation directory.
   - On UNIX, the log file is `$SYBASE/REP-15_2/install/<RepServerName>.log`, where `$SYBASE_REP` points to the Replication Server installation directory.
2. To see information about the connections configured with Replication Server, use **isql** to log in to Replication Server:

   ```
   isql -S <RepServerName> -U <username> -P <password>
   ```

   For example, enter:

   ```
   isql -S SAMPLE_RS -U sa _P sa_pass
   ```

Replication Server Commands

Use Replication Server commands to troubleshoot Replication Server problems.

Command	Action
admin who	Show status of connections
admin who_is_down	Show status of connections that are not running
admin who_is_up	Show status of connections that are running
resume connection to <connection_name>	Resume connection

Command	Action
suspend connection to <connection_name>	Suspend connection
create connection to <connectionname>	Create connection
create replication definition <replication_definition_name>	Create replication definition for table
create function replication definition <replication_definition_name>	Create replication definition for procedure
create subscription <subscription_name for <replication_definition_name> with replicate at <Repconnection_Name>.<database_name>without materialization	Create subscription
trace "on", DSI, DSI_BUF_DUMP	Set trace for Data Server Interface (DSI)
trace "on", SQT,SQT_TRACE_COMMANDS	Set trace for Stable Queue Transaction (SQT)

Using admin who for Your Connection

admin who shows the current status of connections.

This example shows that connection RC25XPEAS.EAS422 is running and waiting for the next event.

```
admin who
go
-----------------
Name         State                  Information
-------------------------           --------------------
DSI EXEC     Awaiting Command       118(1)RC25XPEAS.EAS422
DSI          Awaiting Command       118 RC25XPEAS.EAS422
SQM          Awaiting Message       118: RC25XPEAS.EAS422
```

Changing Connection Grouping Mode

If you change the connection from individual to group messages on the Inbound Message Grouping Preference tab, suspend and resume the connection in Replication Server before the change takes effect in RepConnector.

Restarting Components and Connections

When you are troubleshooting, you may want to restart all of the Replication Server and RepConnector components: Replication Server, Application Server, and the RepConnector connection.

Suspend and resume all connections. Restarting the components may address the problem that is preventing a successful connection.

Purging Replication Server Queues

Purge the queue when messages get delayed in Replication Server. See the *Replication Server Reference Manual* commands for purging Replication Server queues.

Freeing Transaction Log Space

Create space in the database transaction log, as RepConnector and Replication Server may not work properly until space is freed. See the *Adaptive Server Enterprise Reference Manual:Commands*.

Verifying Sent Messages to RepConnector

Verify that Replication Server has sent a message to RepConnector.

1. Enter:

   ```
   admin who,sqm
   ```

2. Check the output:
   - First Seg.Block
   - The physical address of the beginning of the queue
   - Last Seg.Block
   - The physical address of the end of the queue
   - Next Read
   - How far the Replication Server has read between the First Seg.Block and Last Seg.Block.

     The Next Read is usually one more than the Last Seg.Block if the Replication Server has read all of the information in that queue. The difference between the First Seg.Block and the Last Seg.Block is the amount of information in the queue in MB. Purging the queue sets the First Seg.Block and the Last Seg.Block to zero.

3. Determine the database ID and the queue type:

   ```
   1> admin who,sqm
   2> go
   ```

4. Put Replication Server into single-user mode:

   ```
   1> sysadmin hibernate_on
   2> go
   ```

5. If **hibernate** does not work, shut down Replication Server and restart it using the `-M` command (single-user).

6. Purge the queue:

   ```
   1> sysadmin sqm_purge_queue,106,0
   2> go
   1> admin who,sqm
   2> go
   ```

   In this example, the database ID is 106, and the outbound queue is always 0.

7. Turn **hibernate** off:

```
1> sysadmin hibernate_off
2> go
```

Index

A

C

D

E

F

G

I

J

L

M

O

P

Q

R

S

T

U

V

W

X