SYBASE®

Configuration and Users Guide

RepConnector™

15.0.1

[HP-UX]

DOCUMENT ID: DC00769-01-1501-01

LAST REVISED: November 2007

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

Contents

About This Book

Audience

This book is intended for system administrators and database administrators.

How to use this book

This manual contains the following chapters:

- Chapter 1, “Overview,” introduces RepConnector and how it integrates with your systems.
- Chapter 2, “Overview of RepConnector Configuration,” describes how to configure RepConnector.
- Chapter 3, “Configuring Replication Server for RepConnector,” describes how to configure Replication Server® to communicate with RepConnector.
- Chapter 4, “Getting Started with RepConnector Manager,” describes how to begin using RepConnector Manager, the graphic user interface (GUI) for creating, configuring, and managing RepConnector connections.
- Chapter 5, “Configuring RepConnector,” describes how to configure a RepConnector environment.
- Chapter 6, “Managing RepConnector Connections,” describes how to manage a RepConnector connection.
- Chapter 8, “Using the ratool Utility,” describes how to use ratool—the command line utility—to create, configure, and manage RepConnector connections.
- Chapter 7, “Customizing the Sender and Formatter Processors,” describes how to use Unwired Orchestrator with RepConnector.
- “Customizing the Message Generator for TIBCO AECM,” describes how to use the RepConnector API to create a customized message generator the works with RepConnector.
- Appendix A, “Configuration Worksheets,” contains worksheets that can assist you while you configure a RepConnector environment.
- Appendix B, “Troubleshooting,” describes how to troubleshoot the RepConnector environment.

Related documents

Documentation sets for Adaptive Server® Enterprise, Replication Server, and EAServer are included with this product.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

This manual uses these style conventions:

- Commands that you enter exactly as shown appear in a Courier font.

```
setenv SYBASE /work/sybase
```

- Words you replace with the appropriate value for your installation appear in italics.

```
isql -Uyour_username -Pyour_password
```

- The names of files, volumes, and directories appear in italics:

  /work/sybase

- The names of programs, utilities, stored procedures, databases, and commands appear in a sans serif font:

  ratool

- Items on a menu appear with vertical bars showing the menu hierarchy:

  File | Print

Syntax conventions

Syntax formatting conventions are summarized as follows. Examples that combine these elements follow the table.

Key	Definition
variable	Variables (words that stand for values that you fill in) appear in italics.
{ }	Curly braces mean that you must choose at least one of the enclosed options. Do not include braces in the command.
[ ]	Brackets mean that you may choose or omit enclosed options. Do not include brackets in the command.
|	Vertical bars mean that you may choose no more than one option, which must be enclosed in braces or brackets.
,	Commas mean that you may choose as many options as you need. Options must be enclosed in braces or brackets. Separate your choices with commas. Commas may also be required in other syntax contexts.
( )	Parentheses are typed as part of the command.
...	An ellipsis means that you may repeat the last unit as many times as necessary.

Required choices

- Curly braces and vertical bars – choose one and only one option.

  {red | yellow | blue}

- Curly braces and commas – choose one or more options. If you choose more than one, separate your choices with commas.

  {cash, check, credit}

Optional choices

- One item in square brackets – choose or omit it.

```
[anchovies]
```

- Square brackets and vertical bars – choose none or only one.

```
[beans | rice | sweet_potatoes]
```

- Square brackets and commas – choose none, one, or more options. If you choose more than one, separate your choices with commas.

```
[extra_cheese, avocados, sour_cream]
```

Repeated elements

An ellipsis (...) means that you may repeat the last unit as many times as necessary. For example, when you use the alter function replication definition command, you can list one or more parameters and their datatypes for the add clause or the add searchable parameters clause:

```
alter function replication definition function_rep_def
    {deliver as 'proc_name' |
    add @parameter datatype [, @parameter datatype]... |
    add searchable parameters @parameter [, @parameter]... |
    send standby {all | replication definition}
    parameters}
```

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Sybase RepConnector 15.0.1 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and Mixed Case Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

Section 508 compliance statement for RepConnector, Sybase Accessibility at http://www.sybase.com/accessibility.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Overview

This chapter includes general information about Sybase RepConnector.

Introduction

With RepConnector, there is no need to poll a database for changes, or to add triggers to notify of events.

Configuring RepConnector requires the configuration of other servers and software in your RepConnector environment.

RepConnector provides adapters for sending to TIBCO, IBM MQ, SonicMQ, or any J2EE-compliant JMS messaging system. It also allows you to send messages to virtually any other user or application.

RepConnector:

- Delivers database events and metadata from Replication Server to the configured destination.
- Follows transactional behavior
- Manages connections using
  - RepConnector Manager, which is a GUI in the Eclipse framework
  - ratool, which is a command line utility
- Groups database events into a single transaction
- Supports text and image datatypes
- Parses replication events and generates XML documents

- Shares a queue among multiple RepConnnectors
- Transforms incoming database events into XML messages, and routes them into configured message queues
- Transforms incoming database events into their application-specific format
- Routes incoming database events to any destination
- Detects message events and routes them to database tables
- Supports EAServer 5.2 and WebLogic 8.1 application servers

RepConnector architecture

RepConnector is designed based on the JCA (Java Connector Architecture) specification of J2EE. It runs in a J2EE-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 messaging 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 want to send a message to the database, RepConnector transforms the event to a SQL statement.
- Event Sender – routes the event to a messaging system, or to a database for reverse direction. Alternatively, you can customize the module to add a customized message sender plug-in.

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 in either SQL commands or an XML representation of SQL commands.

RepConnector process flow

RepConnector participates in two different process flows:

- Routing database events from Replication Server to messaging systems
- Routing events from messaging systems to database tables

Routing database events from Replication Server to messaging systems

RepConnector routes database events from Replication Server to a messaging system:

1 When an event occurs in the database, Replication Server detects the event and pushes it to the RepConnector 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. See Chapter 7, “Customizing the Sender and Formatter Processors” for more information.

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. See Chapter 7, “Customizing the Sender and Formatter Processors,” for more information.

Routing events from messaging systems to database tables

RepConnector routes events from messaging systems to database tables:

1 The Event Capture module listens for messages (in either a standard SQL format, or in XML) arriving in the configured messaging system.

2 When a message arrives, the Event Capture module receives the message and triggers the Event Transformation Module.

3 The Event Transformation module analyzes the message and transforms it to SQL format, if necessary.

4 After the message is transformed to SQL, the Message Sender module sends the SQL statement to the database table.

Transaction rollback

In the event of failure, RepConnector offers transaction rollback through atomic event processing:

- At any point of failure, the transaction is rolled back.
- RepConnector logs any messages in the log file and stops processing any new events.
- After you fix the failure, the rolled-back events are reprocessed. This guarantees that RepConnector does not lose a transaction.

Status and error reporting

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

CHAPTER 2

Overview of RepConnector Configuration

RepConnector's components include Sybase products as well as third-party products. You must configure each component before you can create RepConnector connections.

Basic steps

The following basic steps are required to configure the RepConnector environment:

RepConnector components include Sybase products as well as third-party products. You must configure each component before you can create RepConnector connections.

1 Set up your database server and Replication Server to send replicated events to RepConnector.

   See Chapter 3, "Configuring Replication Server for RepConnector."

2 Configure your messaging system in the RepConnector environment.

3 Create and configure your RepConnector connection.

   See Chapter 4, "Getting Started with RepConnector Manager," and Chapter 2, "Overview of RepConnector Configuration."

4 Manage runtime control.
   See Chapter 6, "Managing RepConnector Connections."

CHAPTER 3

Configuring Replication Server for RepConnector

This chapter describes how to set up a Replication Server to replicate to RepConnector.

You or your system administrator must establish the RepConnector connection in Replication Server before configuring RepConnector.

This chapter assumes that you have already configured a Replication Server environment, have added the primary database to the replication system (including updating the interfaces file with the connection information for the database server), and have marked the primary tables and procedures for replication. If you have not completed these tasks, see the *Replication Server Configuration Guide*.

To configure Replication Server to replicate to RepConnector:

1 Add an entry for RepConnector in the Replication Server interfaces file.

2 Verify that Replication Server is up and running.

3 Create a database connection in Replication Server to communicate with RepConnector.

4 Create a replication definition in Replication Server to identify the data to be replicated.

5 Create a subscription in Replication Server to identify the location to which the data will be replicated.

6 Resume the database connection.

As you work through this chapter, use the worksheet in Appendix A, Configuration Worksheets, to record the values used to configure the RepConnector connection to Replication Server.

Updating the interfaces file

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

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

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

  This is also the RepConnector connection's DSI name, which you will need later when you configure the RepConnector connection in Chapter 5.

  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. This value should be recorded on line 3.b of the worksheet.

  You can use either the Transmission Control Protocol (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. This value should be recorded on line 3.d of the worksheet.

Adding a RepConnector entry to the interfaces file

To add the interfaces entry, use dsedit, a utility that is part of the Replication Server installation in the *OCS-15_0/bin* subdirectory.

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

See the *Adaptive Server Utility Guide* for more information about the dsedit utility and editing interfaces files.

After you update the interfaces file, view the file to verify that your entry is correct.

The Replication Server interfaces file is *$SYBASE/interfaces*, where *$SYBASE* is the location of the Replication Server installation.

```
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.

Example

This is an interface entry for the RepConnector connection:

```
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.

Verifying that Replication Server is running

Verify the status of your Replication Server:

- Use the Replication Manager plug-in to Sybase Central or WorkSpace. See the Replication Manager help for instructions.
- Use the isql utility to log in to Replication Server. If the login succeeds, you know the server is running. The isql utility is in the Replication Server installation directory in *OCS-15_0/bin*. See the Adaptive Server *Utility Guide* for more information about isql.

  To log in to Replication Server and verify that it is up and running, at the command line enter:

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

  where:

  - *user* 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.

If Replication Server is not up and running, start Replication Server. See the *Replication Server Administration Guide*.

Creating the connection to RepConnector

The following procedure adds the RepConnector connection to the replication system and sets the configuration parameters for this connection.

Before you can create the connection, gather the following information:

- The location of the isql utility, which is in the Replication Server installation directory under *OCS-15/bin*
- 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)

❖ Creating a new RepConnector connection in Replication Server

1 Log in to the Replication Server with a user ID that has system administrator permission:

```
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 server 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'
with dsi_suspended
```

where:

- *dataserver* is the RepConnector connection's 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; for example, `RepConnector`.
- *database* is the name of the database to which you are replicating. Record this value in 3.j of 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 just 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, whose destinations 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.

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 <command> and dsi_xact_group_size parameters, 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 batch to 'off'
set dsi_xact_group_size to '-1'
with dsi_suspended
```

Creating the replication definition

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. If you have already defined a replication definition, you can skip the following procedure.

If you have not already done so, you must mark primary tables or stored procedures for replication before continuing.

❖ Creating a table replication definition in Replication Server

1 Gather the following information and record it on your worksheet where appropriate:

- Primary data server name (in 3.k)
- Primary database name (in 3.j)

- Table names and column field name(s)

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 the create replication definition command, see the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

❖ Creating a function replication definition

To create a function replication definition in Replication Server:

1 Gather the following information and record it on your worksheet, in Appendix A, "Configuration Worksheets," where appropriate:

- Primary data server name (in line 3.k)
- Primary database name (in line 3.j)
- Procedure and parameter name(s)

2 Create the function replication definition:

```
create function replication definition
<relication_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 the command create function replication definition, see the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

Creating and verifying the subscription

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.

You must verify that the subscription is valid *both* at the primary table and at the RepConnector connection.

❖ Creating and validating the subscription at Replication Server

Create the subscription using the RepConnector connection name as the parameter value for with replicate at. This value is the name of the connection you created in the previous section.

1 Gather the following information and record it on your worksheet where appropriate:

- Name of the RepConnector connection (3.a) you created in "Creating the connection to RepConnector" on page 10
- Name of the replication definition

2 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.

Because a RepConnector connection is the destination instead of an actual database, Sybase recommends 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
```

3 Verify that the subscription is valid at the primary database and at 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 (3a).
- *database* is the name of the database to which you are replicating. However, RepConnector does not use the database name; this is a placeholder 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, you must resume the connection from Replication Server to RepConnector.

1 Gather this information:

- The name of the RepConnector connection

- The database name you used when you created the connection at Replication Server

2 To resume a database connection from the command line, enter:

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

where:

- *dataserver* is the name of the connection you have created in previous steps.
- *database* is the name of the database to which you are replicating. This is the same value you used when you created the connection in "Creating the connection to RepConnector" on page 10.

For example:

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

For information about resuming a connection in Sybase Central Replication Manager, see the Replication Server plug-in and Replication Manager documentation and online help.

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

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

CHAPTER 4

Getting Started with RepConnector Manager

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

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

Starting RepConnector Manager

Navigate to the RepConnector Manager installation directory.

For example, enter:

```
cd /opt/sybase/EAServer/repra/eclipse
RepConnectorManager.sh
```

When you invoke RepConnector Manager, the Eclipse workbench opens and displays the Sybase RepConnector Manager Welcome page.

❖ Testing and deploying RepConnector Manager to Unwired Orchestrator

If the Welcome Page does not appear, select Help | Welcome | Unwired Orchestrator.

1 From the Unwired Orchestrator Welcome Page, select Database Event Management. The Replication Roadmap appears.

2 Follow the instructions in the Replication Roadmap to display the Sybase RepConnector Manager view.

❖ **Displaying the Sybase RepConnector Manager view from Eclipse**

1 From the Eclipse menu bar, select Windows | Show View | Other.

2 In the Show View dialog box, select Sybase/RepConnector Manager.

3 Select Sybase RepConnector Manager.

4 Resize or relocate the RepConnector tree view within the Eclipse workbench, if needed.

Managing connection profiles

A RepConnector profile contains the connection property information needed to connect to a RepConnector runtime instance. the profile manages all the configured RepConnector connections defined for an installed RepConnector instance running on an application server. You can configure as many RepConnector profiles as necessary, to manage local or remote RepConnector installations from one RepConnector Manager installation.

When you install RepConnector Manager, the installation provides two sample default profiles:

- `EAServer:8080` for Sybase EAServer
- `WebLogic:7001` for BEA WebLogic Server

❖ **Creating a new profile**

1 Right-click the Sybase RepConnector folder and select New Connection Profile.

2 When the New RepConnector Profile dialog box displays, enter:

- Profile Name – the name of the RepConnector profile. This name must be unique within this instance of RepConnector Manager. The name of the profile can contain alphanumeric characters. It cannot contain any white space. The default is `localhost:8080`.
- Host – the HTTP host name of the machine where the target RepConnector runtime is running. The default is `localhost`.

- Port – the HTTP port number where the target RepConnector runtime is listening. The default is `8080` for EAServer and `7001` for BEA Weblogic Server.
- User – the RepConnector administrator user ID to connect to the RepConnector runtime. The default is `repraadmin`.

Note See "Setting up the RepConnector runtime administrator login" on page 22 for information about changing the default user name and password.

For example, enter:

```
Profile Name: EAServer:8080
Host: myhost
Port: 8080
User: rcuser1
```

3 Select OK to create a new profile, or Cancel to cancel the action.

❖ Renaming a profile

1 Right-click the profile and select Rename Profile.

2 Enter a new profile name that is unique within the same RepConnector Manager instance.

3 Select OK to rename the profile with a new name, or Cancel to cancel the action.

❖ Editing the profile properties

1 Right-click the profile and select Edit Profile Properties.

2 Modify the fields as needed.

3 Select OK to save the new properties, or Cancel to cancel the operation.

❖ Deleting a profile

1 Right-click the profile and select Delete Profile.

2 Select OK to delete the profile from the tree view, or Cancel to cancel the action.

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

❖ Setting up the RepConnector runtime administrator login

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

- The login name and the password must be alphanumeric.
- The length of the password must be 30 or fewer characters.

To modify the login:

1 Navigate to the RepConnector runtime installation location’s *bin* directory.

For example, enter:

```
cd /opt/sybase/EAServer/repra/bin
```

2 Enter:

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

❖ Logging in to the RepConnector runtime

1 Right-click the login profile, then select Login.

2 Expand the profile folder to view the list of configured RepConnector connections. By default, a “sample_repconnector” is installed.

A pop-up box appears, displaying all the RepConnector properties and an empty password field. Enter your password and select Login.

If you have successfully logged in, you can see the connections configured with the RepConnector runtime.

❖ Refreshing the profile

You can refresh the tree view under the profile folder any time the profile folder is connected to the RepConnector runtime. After you refresh the profile, the profile tree view shows the current status of the runtime.

1 Right-click the profile.

2 Select Refresh View from the profile folder.

❖ Logging out of the profile and the RepConnector runtime

When you log out of the RepConnector profile, you disconnect from the RepConnector runtime, and the tree view of the profile folder collapses to a single folder object. You can log out of the RepConnector runtime from the profile only if the profile is connected to the runtime.

1 Right-click the connection profile.

2 Select Logout.

3 Select OK, or select Cancel to cancel the action.

CHAPTER 5

Configuring RepConnector

This chapter describes using RepConnector Manager to create and configure RepConnector connections. See the Eclipse online help for more information about Eclipse.

To use this section, you must have already started RepConnector Manager, created a connection profile, and connected to a RepConnector runtime instance.

Before configuring connections

You can create a RepConnector connection to:

- listen for events from a database and then route those events to a messaging system
- listen for events from a messaging system and then route the events to a database

Inbound and outbound types

To configure a RepConnector connection to listen for events from a database and then route the events to a messaging system, select REPLICATION as the *inbound type* and one of the messaging system (JMS, TIBCO, IBMMQ) as the *outbound type*. The inbound type is the source from which the RepConnector Connection will receive data. The outbound type is the destination to which the RepConnector connection routes the data.

To configure a RepConnector connection that listens for events from a database and routes the events to a user-defined sender processor (a file, for example), select REPLICATION as the inbound type and CUSTOM as the outbound type.

To configure a RepConnector connection to listen for events from a messaging system and then route the events to a database, select one of the messaging system (JMS, TIBCO, IBM MQ) as the inbound type and select DATABASE as the outbound type.

Before you can begin configuring a RepConnector connection, ensure that you have taken these steps:

1 Check the requirements for your RepConnector installation. See Chapter 2 in the *RepConnector 15.0 Installation Guide*.

2 Install RepConnector Manager and verified the installation. See Chapter 2 in the *RepConnector 15.0 Installation Guide*.

3 Complete the post-installation tasks in Chapter 3 of the *RepConnector 15.0 Installation Guide*.

4 Start your application server and set up the messaging system.

Configuring the RepConnector

Before you can validate your new RepConnector connection, you must configure your RepConnector environment and restart your application server.

You can skip this step if you are using EAServer JMS.

If you are configuring:

- A JMS Messaging System, go to “Configuring RepConnector for JMS messaging systems” on page 27.
- A TIBCO Messaging System, see “Configuring RepConnector for TIBCO” on page 28.
- An IBM WebSphere MQ Messaging System, see “Configuring RepConnector for IBM WebSphere MQ” on page 30.
- For routing events from messaging to database, see “Configuring RepConnector for your database” on page 31.

- A customized plug-in, see "Configuring the environment for a custom sender or formatter" on page 32.

Note To set the environment for any messaging system, you must modify *$REPRA_HOME/bin/repra_env.sh* where *$REPRA_HOME* is the installation directory for RepConnector.

Configuring RepConnector for JMS messaging systems

Use this section to set up the RepConnector environment, so that RepConnector can communicate with:

- WebLogic Server (WLS) JMS
- SonicMQ JMS

Sybase recommends that you configure your environment before you proceed with configuring the RepConnector connection. This enables you to ping the JMS messaging system during configuration to verify that it is configured correctly.

❖ Configuring WebLogic Server JMS

If you are using RepConnector on EAServer and you are configuring a RepConnector connection to send events to a WebLogic JMS server, modify the RepConnector batch or script file to configure a connection to a WebLogic server using JMS queues.

Note No additional steps are required if you are running RepConnector on WebLogic Server.

1 Verify that the lines in *repra_env.sh* that add the *weblogic.jar* library file to the CLASSPATH/BOOTCLASSPATH variable setting are *not* commented out and are correct for your environment:

```
/work/bea/weblogic81/server/lib/weblogic.jar:$CLASSPATH
/work/bea/weblogic81/server/lib/weblogic.jar:$BOOTCLASSPATH
```

2 Restart EAServer.

3 Create a RepConnector connection or, if one already exists, use RepConnector Manager to validate the connection configuration. See "Creating RepConnector connections" on page 33.

❖ Configuring for SonicMQ JMS

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

1 Verify that the line in *repra_env.sh* that defines the SONICMQ_HOME environment variable is *not* commented out and points to the installation location for SonicMQ.

```
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 are correct for your environment.

```
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

This section describes how to set up the RepConnector environment so that RepConnector can communicate with:

- TIBCO RV
- TIBCO RVCM
- TIBCO AECM
- TIBCO Enterprise for JMS

Sybase recommends that you configure your environment before you configure the RepConnector connection. This enables you to ping the TIBCO messaging system during configuration to verify that you have configured it properly.

❖ Configuring TIBCO RV, RVCM

Follow these steps to configure your environment for different versions of TIBCO Rendezvous.

1 Verify that the line in *repra_env.sh* that defines the TIBCO_HOME environment variable is *not* commented out and points to the installation location for the TIBCO Rendezvous.

```
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 are correct for your environment.

```
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

1 Verify that the line in *repra_env.sh* that defines the TIBCO_HOME environment variable is *not* commented out and points to the installation location for TIBCO Active Enterprise.

```
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 are correct for your environment.

```
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

1 Verify that the lines in *repra_env.sh* that define the TIBCO_HOME environment variable are *not* commented out and point to the installation location for the TIBCO Enterprise for JMS environment.

```
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 are defined correctly for your environment.

```
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

Sybase recommends that you configure your environment before you configure the RepConnector connection. This enables you to ping the IBM WebSphere MQ messaging system during configuration to verify that it is configured correctly.

❖ Configuring IBM WebSphere MQ

1 Verify that the lines in *repra_env.sh* that define the IBMMQ_HOME environment variable are *not* commented out and point to the installation location for the IBM WebSphere MQ environment.

```
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 are defined correctly for your environment.

```
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 *repra_env.sh*. If the environment variable is not defined correctly, modify it as follows:

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.

Note To configure EAServer, see the EAServer documentation.

❖ Configuring MQ JMS

1 Verify that the line in *repra_env.sh* that defines the IBM MQ_HOME environment variable is *not* commented out and points to the installation location for the IBM WebSphere MQ JMS.

```
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 are defined correctly for your environment.

```
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 *repra_env.sh*. If the environment variable is not defined correctly, modify it as follows:

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

Sybase recommends that you configure your environment before you configure the RepConnector connection. This enables you to ping the database during configuration to verify that the connection is configured correctly.

Note There are no additional steps required for RepConnector to communicate with Sybase Adaptive Server.

❖ Configuring for an Oracle database

1 Verify that the lines in *repra_env.sh* that define the CLASSPATH/BOOTCLASSPATH environment variable are *not* commented out and point to the installation location for your database environment.

```
CLASSPATH= /work/oracle/ora92/jdbc/lib/ojdbc.jar:$CLASSPATH
BOOTCLASSPATH=
/work/oracle/ora92/jdbc/lib/ojdbc14.jar:$BOOTCLASSPATH
```

2 Restart your application server.

Configuring the environment for a custom sender or formatter

Sybase recommends that you configure your environment before you proceed with configuring the RepConnector connection.

See Chapter 7, "Customizing the Sender and Formatter Processors," for more detailed information about using the custom sender and formatter features.

❖ Configuring an EAServer for a custom sender or formatter processor

If you are running RepConnector on EAServer and plan to use a customized sender or formatter processor, define the full path name to the *jar* file containing the customized sender processor or message formatter class or classes for the Java Classes property in the EAServer server configuration property:

1 Start EAServer Manager *jagmgr*, which resides under the EAServer installation *bin* directory.

2 Select Jaguar Manager.

3 Select Tools | Connect.

4 Enter the user name, password, host name, and port number to connect to your EAServer.

   By default, the user name is "jagadmin" with no password, the host name is "localhost" and the port number is 9000.

5 Click Connect to connect to EAServer.

6 Select Servers to expand the folder.

7 Right-click Jaguar and select Server Properties. The Server Properties window appears.

8 Select the Java Classes tab.

9 Click Add to add the SenderProcessor/Formatter *jar* file. A row is added to the Java Classes.

10 Enter the full path to the *jar* file containing the sender processor/message formatter classes.

   For example, at the command line, enter:

```
/work/sybase/EAServer/repra/sample/client/sample.jar
```

11 Click OK to save the value.

12 Restart your application server.

❖ Configuring a WebLogic Server for a custom sender processor or custom formatter

1 Verify that the CLASSPATH variable setting in the *repra_env.sh* includes the full path of the *jar* file containing the customized sender processor or message formatter classes.

```
CLASSPATH=/work/bea/repra/sample/client/sample.jar:$CLASSPATH
```

2 Restart your Weblogic Server.

Creating RepConnector connections

❖ Adding and configuring a new connection

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

1 Right-click the connection profile.

For example, right-click the EAServer:8080 connection profile to create a RepConnector connection defined by the EAServer:8080 connection profile.

2 Select Add a New Connection.

The New Connection wizard starts and the Create a New Connection page appears.

3 On the Create a New Connection page:

a Enter a unique name in the Connection Name field. 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 type options are available.

Select one of the following 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, or IBMMQ 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 inbound type, you can select only DATABASE as the outbound type.
- IBMMQ – 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 must select DATABASE as the outbound type.

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

Select one of the following outbound targets:

- 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.

4 Click Next.

5 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 host name is "localhost."
- *<port_number>* is the port number of the target. The default port number is 8080.

The default URL is:

If the default information is incorrect, change *localhost* to the host name of the target server's HTTP listener and *8080* to the port number on which the target server's HTTP listener is listening.

For example, if the host name is "mymachine," which is 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 the *<AppServer>/repra/logs* directory. The default logging level is INFO.

Choose:

- FATAL – to see information about very 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 the progress of the application 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 might want to start the connection automatically when the application server starts because you need 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 have established a successful connection, you can change the connection properties so that the connection starts automatically.

Note When you start a RepConnector connection, Replication Server attempts to connect to it. When a RepConnector connection is stopped, suspend the connection at Replication Server because Replication Server continually attempts to connect to it, even though it 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 Chapter 7, "Customizing the Sender and Formatter Processors," for information about how creating a user-defined message formatter.

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

6 Click Next.

7 The next page that appears depends on what you selected for inbound type on the Create a New Connection page.

If you selected:

- REPLICATION, the Replication Server Inbound Information page appears. See "Configuring replication information for REPLICATION inbound types" on page 38.
- JMS, the JMS Information wizard page is displayed. See "Configuring JMS information" on page 40.

- TIBCO, the TIBCO Messaging Information wizard page is displayed. See “Configuring TIBCO information” on page 43.
- IBMMQ, the MQ Messaging Information wizard page is displayed. See “Configuring IBM WebSphere MQ Information” on page 45.

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

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

Note You must update the *repra_env.sh* file to include the messaging system’s libraries in the environment and restart your application server before you can use the ping. See “Configuring the RepConnector” on page 26 for more information.

Click OK to exit the status window.

9 Click Next.

10 The next page displayed depends on what you selected for outbound type on the Create a New Connection page.

If you selected one of these message queues or topics:

- JMS – the JMS Information page appears. See “Configuring JMS information” on page 40.
- TIBCO RV or RVCM – the TIBCO Messaging Information page displays. See “Configuring TIBCO information” on page 43.
- TIBCO AECM – the TIBCO Messaging Information page displays. See “Configuring TIBCO information” on page 43.
- IBM WebSphere MQ – the MQ Messaging Information page displays. See “Configuring IBM WebSphere MQ Information” on page 45.
- CUSTOM, and selected Custom Plug-in Class in the General Connection Information wizard page – the Plug-in wizard page displays. See “Configuring custom plug-in information” on page 46.
- DATABASE – the Database Connection Information appears. See “Configuring database connection information” on page 47.

11 Click Next. The Summary page wizard page appears.

The Summary page appears, which summarizes the values you entered for the new connection. These values are saved in a properties file on your local machine called *connection_name.props*, where *connection_name* is the name of the connection. This file resides in the *repra/conf* directory in your application server installation directory structure.

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

To change the 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.

Configuring replication information for REPLICATION inbound types

You must enter the Replication Server inbound information and the Replication Server System Database information if your inbound type is REPLICATION.

Configuring Replication Server inbound information

Follow these steps to define the name of the Data Server Interface (DSI) and the port number that the RepConnector connection will listen on, along with the user ID and password use for Replication Server to connect to RepConnector connection.

The values required for this procedure can be found in Appendix A, "Configuration Worksheets" and are designated in parenthesis in the steps that follow.

❖ Entering Replication Server inbound information

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.

Return to "Adding and configuring a new connection" on page 33.

Configuring Replication Server System Database (RSSD) information

This section describes how to define the information that RepConnector uses to connect to the Replication Server System Database (RSSD). RepConnector uses metadata information from the RSSD to process the events received from Replication Server.

❖ Entering Replication Server System Database information

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. With large text or image data groupings, your system may run out of memory after accumulating only several messages.

Return to step 10 in "Adding and configuring a new connection" on page 33.

Configuring JMS information

Before you configure your JMS queue, verify that your application server is started. If you are using EAServer, verify that the message service is configured.

If your connection is to a JMS message queue or topic, enter the following information on the JMS Information page of the Create Connection wizard.

1 Choose the destination type:

- Queue – to use point-to-point messaging.
- Topic – to use publish and subscribe messaging.

  See your JMS documentation for more information about destination type.

2 Enter the JMS provider URL. This URL is the host name and port number that will be used to connect to the JMS Server.

- If you are using EAServer JMS, enter:

  iiop://<host_name >:port_number

  where:

  - *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.

  For example: `iiop://my_machine:9000`

  If you are using WebLogic Server JMS, the protocol type must be "t3", which is the WebLogic multitier JDBC driver.

  where:

- *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:

```
t3://<host_name>:<port_number>
```

For example: `t3://mymachine:7001`

- If you are using TIBCO JMS or SonicMQ JMS, enter:

```
tcp://<host name>:<port number>
```

where:

- *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.

For example: `tcp://localhost:7222`

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

- If you are using EAServer JMS, select or enter:

```
com.sybase.jms.InitialContextFactory
```

- If you are using WebLogic Server JMS:

```
weblogic.jndi.WLInitialContextFactory
```

- If you are using TIBCO JMS:

```
com.tibco.tibjms.naming.TibjmsInitialContextFactory
```

- For 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 Select or enter the name of the connection factory administered object.

- If you are using EAServer JMS and the destination type is a queue:

```
javax.jms.QueueConnectionFactory
```

If destination type is a topic:

```
javax.jms.TopicConnectionFactory
```

- If you are using WebLogic Server JMS and the destination type is a queue:

  `weblogic.jms.QueueConnectionFactory`

  If destination type is a topic:

  `weblogic.jms.TopicConnectionFactory`

  Note This name must match the name in the server for the connection factory administered object. Make sure that this connection factory administered object name exists in your WebLogic Server, or change the value here to match the name of the connection factory administered object you have created in WebLogic Server.

- For TIBCO JMS, if the destination type is a queue:

`com.tibco.tibjms.TibjmsQueueConnectionFactory`

If the destination type is a topic:

`com.tibco.tibjms.TibjmsTopicConnectionFactory`

Note This name must match the name in the server for the connection factory administered object. Make sure that this connection factory administered object name exists in your TIBCO JMS Server or change the value here to match the name of the connection factory administered object you have created in TIBCO JMS Server.

- For 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:

`JMS_Queue`

6 Enter the user name and password for the queue or topic, for example, enter `jagadmin` with no password.

7 If you have selected Topic as the destination type, 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.

If you have just configured inbound information, return to step 10 in "Adding and configuring a new connection" on page 33.

Configuring TIBCO information

If you are using a TIBCO messaging system, enter the following information on the TIBCO Messaging Information page of the Create Connection wizard.

If necessary, see your TIBCO documentation for information about the TIBCO product.

1 In the TIBCO Message Type field, choose:

- RV – to configure a TIBCO Rendezvous-reliable message. Proceed to Step 2.
- RVCM – to configure a TIBCO Rendezvous Certified Messaging (RVCM) message. Proceed to Step 2.
- AECM – to configure a TIBCO Rendezvous Active Enterprise Wired Format messaging message. Proceed 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 Enter the TIBCO Rendezvous daemon value:

```
protocol:hostname:port
```

For example, enter `tcp:my_machine:7500`.If you do not specify a value for the host name, it defaults to “localhost.”

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, specify the host name; for example, enter:

```
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.

e If you are using an RVCM message type, enter the name of 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 RVCM message type, enter in the CM Duration field minutes that TIBCO message system stores unread messages in the *cmledger*.

The default value is 0, which means the messaging system keeps messages in the *cmledger* forever.

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

3 If you selected AECM as the TIBCO message type, enter the information for AECM.

- 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 messaging system to database, enter the destination name in the Status Destination field. The status destination queue or topic you define is used for client application to listen an error message (if any) that may result in the event being sent to the database.

If you have just configured inbound information, return to step 10 in "Adding and configuring a new connection" on page 33.

Configuring IBM WebSphere MQ Information

If you are using an IBM WebSphere MQ messaging system, enter the following information on the MQ Messaging Information page of the Create Connection wizard.

1 Choose:

- MQ for IBM WebSphere MQ Messaging System
- MQJMS for IBM WebSphere MQ JMS Messaging System

2 Choose:

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

Select the encoding type you want to use 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 WebSphere 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.`

In the Queue Manager/Factory field, enter the name of the IBM WebSphere MQ queue manager. For example, enter `MQBiz2QM.`

5 In the Queue Name field, enter the name of the IBM WebSphere MQ destination. For example, enter `MQBiz2Queue`.

6 Enter the user name and password to the connect to IBM WebSphere 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 for the destination name defined in Queue Name field.

7 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.

If you have just configured inbound information, return to step 10 in "Adding and configuring a new connection" on page 33.

If you have just configured outbound information, return to step 13 in "Adding and configuring a new connection" on page 33.

Configuring custom plug-in information

If you are using a user-defined message formatter or message sender processor, enter the following information in the Plug-in Information page of the Create Connection wizard.

1 If you have selected Custom Plug-in Class in the General Connection Information wizard page, enter the class name for your customized message formatter. For example, enter:

```
sample.MyMessageFormatter
```

2 Enter the path to the property file associated with this customized message formatter:

```
MyMessageFormatter.prop
```

3 If you have selected CUSTOM as your outbound type,

a Enter the class for your customized message sender processor in the Sender Processor Plug-in Class field:

```
sample.FileClient
```

b Enter the path to the property file information associated with this sender processor plug-in:

```
FileClient.prop
```

4 If you have just configured outbound information, return to step 13 in “Adding and configuring a new connection” on page 33.

Configuring database connection information

If your outbound connection is to a database, enter the following information in the Database Information page of the Create Connection wizard.

See “Configuring RepConnector for your database” on page 31 for more information on how to configure your environment.

1 Enter the JDBC URL information to connect to the database in the JDBC Connection URL field.

For example, to connect to Adaptive Server, enter, where “testmachine” is the name of the machine where the data server is running and 5000 is the port number that the data server is listening on:

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

2 Enter the name of the JDBC driver class that will be used 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 that will be used to connect to the database.

Return to “Adding and configuring a new connection” on page 33.

CHAPTER 6

Managing RepConnector Connections

This chapter describes managing your RepConnector connections with RepConnector Manager.

This chapter assumes you have already started RepConnector Manager and have connected to the RepConnector instance. If you have not, see Chapter 4, “Getting Started with RepConnector Manager.”

Note You can also use the command line utility, ratool, to manage your RepConnector connections. See Chapter 8, “Using the ratool Utility” for more information.

Managing a connection

This section describes how to start, stop, refresh, rename, copy, delete, and validate a connection. It also describes how to view and modify the properties of a connection as well as how to view both the connection log and the runtime log.

❖ Starting a connection

If a RepConnector connection is already started, an attempt to start the connection results in a warning message indicating that the connection is already started.

1 Right-click the connection and select Start Connection.The Start Connection Status pop-up window displays the connection status.

2 Click OK to exit the pop-up window.

If the connection fails to start, see the RepConnector connection log for information about the failure. See “Viewing connection log information” on page 52.

❖ Stopping a connection

If a RepConnector connection is already stopped, an attempt to stop the connection results in a warning message indicating the connection is already stopped.

1 Right-click the connection and select Stop Connection. The Stop Connection pop-up window displays.

2 Click Yes to stop the connection. Click No to cancel the operation.

The Stop Connection Status window displays the connection status.

3 Click OK to exit the pop-up window.

If the connection fails to stop, check the RepConnector connection log for more information. See “Viewing connection log information” on page 52.

❖ Refreshing a connection

This procedure refreshes a running RepConnector connection by reloading the connection properties and restarting the RepConnector connection. If you have a running connection, you can modify the connection properties, and then select Refresh Connection to restart the connection with the newly modified connection properties.

1 Right-click the connection and select Refresh Connection. The Refresh Connection Status pop-up window displays the connection status.

2 Click OK to exit the pop-up window.

If the connection fails to refresh, see the RepConnector connection log for more information. See “Viewing connection log information” on page 52.

❖ Renaming a connection

You can rename an existing connection; however, you must first stop the connection.

1 Right-click the connection and select Rename Connection.

2 Enter the new connection name in the text field.

3 Click OK to exit the Rename Connection window.

❖ Deleting a connection

You can delete an existing connection; however, you must first stop the connection before you can delete it.

1 Right-click the connection and select Delete.

2 Click OK to delete the connection. Click Cancel to cancel the delete operation.

❖ Copying a connection (Save As)

You can 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 in the text field.

3 Click OK to exit the Save As window.

❖ Validating a connection

You can validate your connection configuration information. This procedure validates both the inbound and outbound configuration properties and returns the status of the validation.

1 Right-click the connection and select Validate Connection. The Validation Connection status pop-up window displays the connection status.

2 Click OK to exit the pop-up window.

If the connection fails validation, refer to the RepConnector connection log for more information. See “Viewing connection log information” on page 52.

❖ Viewing or modifying properties for an existing connection

You can view or modify the connection properties for 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 Properties window displays. The left pane contains a tree structure with four categories of connection property information.

- Select General Properties to view general information about the connection.
- Select Inbound Type Properties to view inbound configuration information.

- Select Outbound Type Properties to view outbound configuration information.
- Select User Defined Plug-in Properties if this connection contains a customized plug-in.

2 Modify the values in the right pane.

3 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

1 Right-click the connection and select View Log.

For example, you can right-click sample_repconnector and select View Log.

The View Connection Log window displays.

2 Click the X in the top right corner of the window to exit the View Connection Log window.

Note 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

1 Right-click the connection profile and select View Log.

For example, you can right-click the EAServer:8080 connection profile and select View Log.

The View Runtime Log window displays.

2 Click X on the top right corner of the window to exit the View Connection Log window.

Note 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 on the profile and select View Log.

❖ Refreshing the connection view display

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

1 Right-click the connection profile.

2 Select Refresh View.

The connection view for the RepConnector instance is refreshed.

CHAPTER 7

Customizing the Sender and Formatter Processors

RepConnector allows you to customize the sender processor and the formatter processor for routing the incoming replication events to meet your application needs. To do this, you must:

- Develop your own Java class implementing APIs that are provided by RepConnector
- Define the class in your connection configuration
- Modify the server environment

Note You must indicate that you will be using a customized sender processor when you configure the RepConnector connection. See Chapter 2, “Overview of RepConnector Configuration,” for more information about configuring RepConnector connections.

Customizing the sender processor

To create a customized sender processor that runs within the RepConnector environment:

1 Create a sender processor for your application by implementing the *com.sybase.connector.repra.RepraClient* interface.To use RepConnector to load a property page, implement *com.sybase.connector.repra.RepraCustomClient*.

2 Use the RepConnector Manager Add Connection wizard to create a RepConnector connection that routes events to the custom sender processor.

3 Select Replication as the inbound type when you configure the new connection, and select Custom as the outbound type.

4 Later in the wizard, enter the name of your custom class. If your custom class loads a property page, enter the full path as well as the file name of the property page.

5 Modify the system environment to add the customized sender processor.

   Define the full path of the *jar* file or the directory containing the customized sender processor:

   - For EAServer – the Java classes properties.
   - For WebLogic – CLASSPATH in *repra_env.sh* file.

   For detailed steps, see "Building the runtime environment for the customized formatter processor" on page 62.

6 Shut down and restart the application server. You must restart the application server before accessing the customized sender processor.

RepraClient interface

```
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 of the 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)
        {
        }
}
```

❖ **Compiling the customized sender processor**

1 Change to the directory of your sender processor.

2 Use the Java compiler by defining the -classpath parameter with the required libraries to compile the customized class.

```
cd /work/custom
mkdir customclasses
/usr/jdk141/bin/javac -classpath .:/opt/sybase/EAServer/repra/
lib/repraconn.jar -d customclasses com/mycompany/SampleClient.java
```

3 Verify the compilation.

When the compilation finishes without any error messages, go to the *customclasses* directory to verify that *SampleClient.class* is under *com/mycompany*.

❖ **Building the runtime environment for the customized sender processor**

To use a *jar* file that includes the customized sender processor, go to the *customclasses* directory and use the jar command to build a jar file from the com directory. Otherwise, you can use the directory path to set up your environment.

Note Sybase recommends that you use *jar* file format for the customization.

1 Build a *jar* file.

- For EAServer:
  - a Copy the *jar* file or the directory structure containing the Java classes created from the previous step to the EAServer or *java/classes* directory.
  - b If you are using the *jar* file, start EAServer and connect to it from EAServer Manager.

    Go to the *Servers/<your server>* directory and right-click the Server Properties menu. Select the Java Classes tab and click Add to add the name of the *jar* file, such as *mycustom.jar*. Click OK.
  - c Add *repra.con.jar* to the Server Properties.
- For WebLogic:
  - a Shut down the application server if it is running.
  - b Modify the *$BEA_HOME/repra/bin/repra_env.sh* file to add the full path of *mycustom.jar* or the *customclasses* directory to the end of the CLASSPATH definition.

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

c Start the WebLogic Server to activate the new environment change.

Note For more information, see "Creating new custom sender and custom formatter classes" on page 63.

Customizing the formatter processor

You can create your own customized formatter processor for your application by implementing the interface *com.sybase.connector.repra.rep.RepTransactionFormatter*. This is the same property page option as RepraCustomClient and RepraCustomTransactionFormatter, but you can use these new interfaces to create a customized formatter processor that also loads a custom property page.

1 Use the Configure Connection wizard in RepConnector Manager to configure the connection.

2 On the Configure Replication window, set the inbound type to REPLICATION.

3 On the General Properties window, select Customized Plug-in Class.

4 Modify the system environment to add the customized formatter processor.

   You must define the full path of the *jar* file or the directory containing the customized formatter processor:

   - For EAServer – Java classes properties for EAServer.
   - For WebLogic – CLASSPATH in *repra_env.sh*.

5 Shut down and restart the application server.

RepTransactionFormatter interface

```
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 under *<AppServer location>/repra/sample/client* directory as a sample of the customized message formatter. The sample uses the DBEventParser utility to retrieve data and metadata from the replication event. For detailed information see “Using the DBEventParserFactory” on page 64.

❖ Compiling the customized formatter processor

1 Change to the directory where your formatter processor is located.

```
cd /work/custom
```

2 Use the Java compiler, the -classpath parameter, and the required libraries to compile your customized class.

```
 mkdir customclasses
/usr/jdk141/bin/javac -classpath.:/opt/sybase/EAServer/repra/
lib/repraconn.jar -d customclasses com/mycompany/SampleFormatter.java
```

3 Verify the compilation.

When the compilation finishes without any error messages, go to the *customclasses* directory to verify that the *SampleFormatter.class* exists in *com/mycompany*.

❖ Building the runtime environment for the customized formatter processor

To include the customized formatter processor in a *jar* file, go to the *customclasses* directory and build a *jar* format from the *com* directory. Otherwise, you can use the directory path for setting up your environment.

Note Sybase recommends that you use the *jar* file format for your customization.

---

1 Build a *jar* file:

```
cd /work/custom/customclasses
/usr/jdk141/bin/jar -cf mycustom.jar com
```

2 Shut down and restart your application server.

For EAServer:

1 Copy the *jar* file, or the directory structure containing the Java classes created from the previous step, to the EAServer *java/classes* directory.

2 Run EAServer and connect to it from EAServer Manager.

3 If you are using the *jar* file, go to the *Servers/<your server>* directory and select Server Properties. Select the Java Classes tab. Click Add to add the name of the *jar* file, such as *mycustom.jar*, and click OK.

4 Shut down the EAServer, and restart it to activate the new environment change.

For WebLogic:

1 Shut down the application server if it is running.

2 Add the full path of *mycustom.jar* or the *customclasses* directory to the end of the CLASSPATH definition in *$BEA_HOME/repra/bin/repra_env.sh*:

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

3 Start the WebLogic Server to activate the new environment change.

Creating new custom sender and custom formatter classes

To create a user-defined property page, implement two new interfaces, RepraCustomClient and RepraCustomTransactionFormatter. Both add the same two new methods to the methods already implemented by RepraClient and RepTransactionFormatter:

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

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

These new interfaces provide two new samples, *CustomMessageFormatter.java*, and *MailClientCustom.java*, to the RepConnector installation's <*sample/client*> directory. MailClientCustom uses a custom configuration file named *sender.props*, also in the sample directory.

Using the *DBEventParserFactory*

RepConnector provides a utility called DBEventParserFactory that you can use to extract both the metadata and the actual data from a single or grouped replication event. This utility is intended to be used with the customized formatter for extracting and reformatting data before sending it to the destination. This utility can also be used by an end-user application that has received the XML representation of the event.

To obtain a parser instance, enter:

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

DBEventParser APIs

Your customized formatter can use the retrieved information to regenerate a new message in a customized format to send to the sender processor. The following section describes the APIs of the DBEventParser utility.

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

Description

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

Syntax

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

int size()

Description

Returns the number of operations in the transaction.

String getDSName() throws Exception

Description

Returns the DSI name defined for the connection.

String setDBName() throws Exception

Description

Returns the database name defined for the connection.

String getEventId() throws Exception

Description

Returns the unique event ID of this transaction.

String getOperation(int elemAt)

Description

Returns the operation (valid returns are insert, delete, update, and exec) at the position of elemAt. If there is only one element, use 0.

Examples

```
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

Description

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

Examples

```
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

Description

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

Description

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

Description

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

Example

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

Vector getData(int elemAt) throws Exception

Description

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 has the following 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

Description

Returns a Vector containing hash tables which 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 has the following 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

Description

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

int getFieldType(Hashtable field) throws Exception

Description

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

JDBC datatype constant	Constant value (int)
DBEventParser.CHAR	0
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

JDBC datatype constant	Constant value (int)
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
DBEventParser.TIME	28

Object getFieldValue(Hashtable field) throws Exception

Description

Returns the field value of this column as an object.

The subtype of the object is determined by the following mapping:

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

Description

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

Example

```
. . .
System.out.println(toXMLText
    (http://yjeongw2k:8080/RepraWebApp/dtds
    /dbeventStream.xsd));
```

Sample output:

```
<!DOCTYPE dbStream SYSTEM
  'http://yjeongw2k:8080/RepraWebApp/dtds/dbeventstream.xsd'>
<dbStream environment="repraJMS2.repdb">
  <tran eventId=
"102:0000000000000ab200003e10004b00003e1000490000937300dda25000000000000
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>
```

Using the RaXMLBuilder utility

The RaXMLBuilder utility helps user applications that send a message containing database events to RepConnector. This utility generates an XML message format, containing the database events that the user application sends to RepConnector for routing to the database.This section documents the API for this utility.

RaXMLBuilder()

Description

The default constructor.

Syntax

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

createTranDocument() throws Exception

Description

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 name of 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

Description

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 name of 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

Description

Adds the database operation to the current event, either <tran> element or <dbEvent> element. If the event type exists and it already contains an operation, it 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 name of the SQL operation, such as insert, update, delete, exec.

schemaName – The name of the target table.

addValue() throws Exception

Description

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. All of the value must be passed as String.

addInValue() throws Exception

Description

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

Description — 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

Description — 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 – one of the SQL conditions, either AND or OR.

operator – a SQL operator.

write() throws Exception

Description — Prints XML text to the specified file.

Syntax

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

Parameters

filename – the name of the target file.

xmlDocByteArray() throws Exception

Description — Returns a ByteArrayOutputStream containing the XML data.

Syntax

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

xmlDocString() throws Exception

Description — Returns a String containing the XML data.

Syntax

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

cancelOperation() throws Exception

Description — 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

Description — 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

Description — Returns the error code from the error document.

Syntax

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

getErrorMessage() throws Exception

Description — Returns the error message from the error document.

Syntax

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

String getOwner(int elementAt) throws Exception

Description — Retrieves the owner name of the replication event.

Example

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

Configuring the RaXMLBuilder

You must include the *repraconn.jar* file in your CLASSPATH environment variable setting.

- For bsh enter:

  CLASSPATH =$REPRA_HOME/lib/repraconn.jar:$CLASSPATH
  export CLASSPATH

- For csh enter:

  setenv CLASSPATH $REPRA_HOME/lib/repraconn.jar:$CLASSPATH

❖ Using 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 two parameters:

- The command
- The name of the schema

For example:

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

5 Add data to the operation, which requires these parameters:

- 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 these parameters:

- 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

A sample implementation, *UseXMLBuilder.java*, is included in the RepConnector installation <*sample/client* >directory. When you compile and run the sample, include the *repraconn.jar* file in your CLASSPATH. For example, enter:

```
java-classpath.:$REPRA_HOME/lib/repraconn.jar:$REPR_HOME/lib/
dom4j-full.jar UseXMLBuildertext.xml
```

The result looks like the following:

```
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, you can 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

When you build and send XML data to the TIBJMS queue, the following files help you compile and run the sample JMS client.

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

Handling ownership information

The XML utility schema allows you to handle 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 the XML schema you must upgrade the file in the directory *$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.

Example 1

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:8080/RepraWebApp/dtds
/dbeventstream.xsd" environment="RepConn.repdb">
<tran eventId=
"102:0000000000028cfc000007d8000f000007d8000c0000955700c0789e0
00000000001001">
<delete schema=RepTable3">
 <oldValues>
  <cell name="repId" type="INT">1</cell>
 </oldValues
</delete>
</tran>
</dbStream>
```

Example 2 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:8080/RepraWebApp/dtds
/dbeventstream.xsd" environment="RepConn.repdb" owner="dbo">
<tran eventId=
"102:0000000000028cfc000007d8000f000007d8000c0000955700c0789e0
00000000001001">
<delete schema=RepTable3">
 <oldValues>
  <cell name="repId" type="INT">1</cell>
 </oldValues
</delete>
</tran>
</dbStream>
```

CHAPTER 8

Using the *ratool* Utility

The RepConnector command line utility (ratool) provides an alternative to RepConnector Manager.

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

Sybase recommends that you use the RepConnector Manager for all configurations. However, ratool can be useful if you are performing batch processing..

ratool utility

The ratool utility is an alternative way to administer and configure your connection.

Note Command options are case sensitive.

Syntax

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

Options

Table 7-1 shows the valid options of ratool.

Table 8-1: ratool options

Option	Description
-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 8080.
-user *<username>*	The RepConnector administrator login user ID. The default is repraadmin.
password *<password>*	The password for RepConnector administrator login. By default, there is no administrator password.
-help	Displays a usage message. If used alone, help displays help on all ratool commands. If used with the name of a command line flag or command option, help displays help for that command line flag or command option.
help *<command_option>*	Displays the help information for a specific ratool *command_option*. If a *command_option* is not specified, it displays a list of the available command options. **Note** 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 display. Valid values are: • FATAL • ERROR • WARNING • DEBUG • INFO

Option	Description
Command	
-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 -file is specified, this option writes the logging information to a file specified by *logFile*.
-getProperty <*connName*> [-file <*propfile*>]	Displays the connection properties for a specific connection. If -file is specified, this option writes the connection property information to a file specified by *propfile*.
-import <*conn_name*> <*conn_prop_file*> [-override]	Adds a new connection. If you specify -override, this command option updates the connection property information for an existing connection.
-list	Lists all known connections.
-ping <*connName*> <*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 a specific connection.
-rename <*conn_name*> <*new_conn_name*>	Renames a connection name.
-start <*conn_name*>	Starts a specific connection.
-startAll	Starts all connections
-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.
-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.

-copy

Description Copies the connection name.

Syntax ratool -copy <*src_connection_name*> <*dest_connection_name*>

Parameters

src_connection_name
The name of the connection to copy.

dest_connection_name
The name of the connection you are creating from the source connection.

Examples To copy the connection named RepToJMS to a new connection named RepToJMS2:

```
ratool -copy RepToJMS NewRepToJMS
```

Usage Creates a new connection by copying a connection name.

-delete

Description Deletes a specified connection.

Syntax ratool -delete <*connection_name*>

Parameters

connection_name
The name of the connection that you want to delete.

Examples To delete a connection RepToJMS:

```
ratool -delete RepToJMS
```

Usage Deletes one connection.

-getLogInfo

Description Retrieves the connection log information for a specified connection.

Syntax ratool -getLogInfo <*connection_name*> [-file <*log_file*>]

Parameters

connection_name
The name of the connection for which you want log information.

log_file
The name of the log file to which you are sending the connection log information.

Examples

Example 1 To get the connection log information for connection RepToJMS:

```
ratool -getLogInfo RepToJMS
```

Example 2 To get the connection log information for connection RepToJMS and send log information to *RepToJMS.log*:

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

Example 3 To get the connection log information for connection RepToJMS and send log information to the default log file (*defaultRepToJMS.log*):

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

Usage

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

-getProperty

Description

Retrieves the connection property information for a connection.

Syntax

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

Parameters

connection_name
The name of the connection for which you want log information.

props_file
The name of the file to which you are sending the connection property information.

Examples

Example 1 To get the connection property information for connection RepToJMS:

```
ratool -getProperty RepToJMS
```

Example 2 To get the connection property information for connection RepToJMS and send it to *RepToJMS.props*:

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

Example 3 To get the connection property information for connection RepToJMS and send it to the default connection file (*defaultRepToJMS.props*):

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

Usage

The information returned is in the form of 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

Description

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 name of the connection to import.

connection_prop_filename

The name of the file that contains the properties to import.

-override

This option overrides the existing connection information. If you do not specify -override and there is an existing connection, this option returns a failure message.

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

Examples

Example 1 To add a new connection using the properties in *RepToJMS.props*:

```
ratool -import RepToJMS /repraconf/RepToJMS.props
```

Example 2 To update an existing connection using the properties in *RepToJMS.props*:

```
ratool -import RepToJMS /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

Description — Lists all known connections.

Syntax — ratool -list

Examples — To list all connections.

```
ratool -list
```

Usage

-ping

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

Syntax — ratool -ping <*connection_name*> <*ping_type*>

Parameters

connection_name

The name of the connection for which you are verifying the configuration information.

ping_type

The type of connection you are pinging to verify connection 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** To verify 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 To verify the inbound configuration for connection RepToJMS by pinging the inbound device (for example, server or messaging system):

```
ratool -ping RepToJMS INBOUND
```

Example 3 To verify 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

Description — Refreshes a specified connection.

Syntax — ratool -refresh <*connection_name*>

Parameters — *connection_name*
The name of the connection to refresh.

Examples — To refresh 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, this option returns a warning message.

-refreshAll

Description — Refreshes all running connections.

Syntax — ratool -refreshAll

Examples — To refresh 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

Description Renames a connection.

Syntax ratool -rename <*old_connection_name*> <*new_connection_name*>

Parameters

old_connection_name
The name of the connection to rename.

new_connection_name
The new name for the connection.

Examples 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, an error message appears.

-start

Description Starts a specified connection.

Syntax ratool -start <*connection_name*>

Parameters

connection_name
The name of the connection to start.

Examples To start a connection called RepToJMS:

```
ratool -start RepToJMS
```

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

-startAll

Description Starts all the connections.

Syntax ratool -startAll

Examples To start all known connections.

```
ratool -startAll
```

Usage Use -startALL to start connections.

-status

Description — Gets the status of a specific connection.

Syntax — ratool -status [<*connection_name*>]

Parameters — *connection_name*

The name of the connection you want to status for.

Examples — **Example 1** You can use -status with the ratool options to display the status of a variety of parameters; for example, the port name, different passwords, and so forth. For example:

To get the status of RepToJMS:

```
ratool -status RepToJMS
```

Example 2 To get the status of all configured RepConnector connections for RepConnector running on “localhost”, listening on port 8080, connecting as user “repraadmin” with no password:

```
ratool -status
```

Example 3 To display debug logging information while running ratool, issue:

```
ratool -host machine1 -port8888 -user newuser -password newpassword -
loglevel DEBUG -status
```

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

Example 4 If you are connecting to RepCoonnector running the the BEA default 7001 port you can run ratool:

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

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

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

-stop

Description — Stops a specified connection.

Syntax ratool -stop <*connection_name*>

Parameters *connection_name*

The name of the connection to start.

Examples To stop connection RepToJMS:

```
ratool -stop RepToJMS
```

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

-stopAll

Description Stops all running connections.

Syntax ratool -stopAll

Examples To stop all known connections:

```
ratool -stopAll
```

Usage If the connection is not running, this option returns a warning message.

CHAPTER 9

Customizing the Message Generator for TIBCO AECM

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

This chapter describes 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. At the user exit, a place in a software program where a customer can arrange for a user-designed program to be called, you can create your own Java implementation to customize a wire-formatted message.

Configuring properties for RepConnector

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

Connection configuration

Table 9-1 lists the parameters for using the TIBCO Active Enterprise feature and the customized message generator:

Table 9-1: Parameters for TIBCO Active Enterprise

Property name	Description
Inbound Type	The inbound type must be set to REPLICATION. For example: `Inbound Type=REPLICATION`
Outbound Type	The type of sender the client processor uses for sending out messages. In this case, select TIBCO. For example: `Outbound Type=TIBCO`
TIBCO Message Type	The transport type for TIBCO must be selected as AECM.
AE Configuration File	Active Enterprise configuration properties. 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.
AE Message Generator	The Customized Message Generator class name, and Active Enterprise-specific property. For example: `MsgGenerator=sample.MyMsgGenerator`

Property file containing the *Active Enterprise Connection/Customization (ae.props)*

Table 9-2 lists the properties that are required 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 9-2: 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=//Sybase/eas/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 is 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`

Using the base class APIs

This section describes the base class APIs that you can use to build a custom TIBCO AECM message generator.

Default 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. In this case, the default is to put the XML text stream into the data field of the Active Enterprise message and to add it to the M-tree node.

Customized TIBCO AECM message generator

You can 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. You must 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 without any input argument.

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.

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

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

MClassRegistry getClassRegistry()

Description — Returns the current MClassRegistry object.

setClassRegistry(MClassRegistry reg)

Description — Sets the current MClassRegistry with the input object.

String getOwner(int elementAt) throws Exception

Description — 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)

Description — Returns a string value with the given key from the properties file defined as the AE Configuration file of the connection configuration. It returns a null value if the key is not found.

String getProperty(String key, String defValue)

Description — Returns a String value with the given key from the properties file defined AE Configuration file of the connection configuration. It returns the defValue if the key is not found.

setProperties(Properties props)

Description — Sets the current properties with the input Properties object.

Properties getProperties()

Description — Returns the current properties.

MTree createMInstance(Object repmsg) throws Exception

Description — Builds the M-Tree for the TIBAECM client and returns it. The MPublisher sends out this MTree object to MSubscribers.

APIs retrieving 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 "Using the DBEventParserFactory" on page 64 for details about APIs.

Configuring and using the default wire-formatted message generator

Configuring connections

Example

```
Inbound Type = REPLICATION
Outbound Type = TIBCO
TIBCO Message Type = AECM
AE Configuration File = //sybase/eas/conf/ae.props
AE Message Generator =
```

SDK application configuration with sybase/eas/repra/conf/ae.props

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/sim
pleSKD_adapter
remote_repository=Sybase/eas/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_adap
ter
```

Example output

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

Note The following message has been formatted for readability.

```
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:8080/repra/dtds/dbeventstream.dtd'>
      <dbStream environment="repraJMS2.repdb">
            <tran
eventId="102:000000000000c8d500007fe7003900007fe70036000093bd00bf1c0c00000
00000010001">
              <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>"}
}
}
```

Configuring and using the customized wire-formatted message generator

This section has a summarized overview of the different components from the back-end server, Adaptive Server Enterprise, and Replication Server. It explains how to configure the RepConnector to deliver a TIBCO AE message to a TIBCO message bus.

Note This section assumes you have 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=//sybase/eas/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/sim
pleSDK_adapter
remote_repository=F:/EAS 52/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
```

This 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 for an example for the customized AE message generator class.

❖ Compiling the customized message generator

1 Change to the location of your message generator:

```
cd /work/custom
```

2 Use the Java compiler to define the -classpath parameter with the required libraries to compile the customized class. For example:

```
mkdir customclasses<enter>
/usr/jdk141/bin/javac -classpath
.:/opt/sybase/EAServer/repra/lib/repraconn.jar
<enter>
```

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

❖ **Verifying the compilation**

1 If the compilation command finishes without any error messages, go to the *customclasses* directory.

2 Verify that *MyMsgGenerator.class* exists in *com/mycompany*.

3 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

To use a *jar* file including the customized message generator, go to the *customclasses* directory and use the jar command to build a *jar* format from the *com* directory. Otherwise, you can use the directory path to set up your environment.

Note Sybase recommends that you use *jar* file format for the customization.

❖ **Building a *jar* file**

1 Change to the *customclasses* directory:

```
cd /work/custom/customclasses
```

2 Build the *jar* file:

```
/usr/jdk141/bin/jar -cf mycustom.jar com
```

3 Add the path to *mycustom.jar* to your environment.

- For EAServer:

  a Copy the *jar* file or the directory structure containing the Java classes created from the previous step to the *java/classes* directory for EAServer.

  b Run EAServer and connect to it from Jaguar Manager.

  c If you are using the *jar* file, go to the *Servers/<your server>* directory and select the Server Properties menu.Select the Java Classes tab and click Add to add the name of the *jar* file, such as *mycustom.jar*.

  d Click OK.

e Shut down and restart EAServer to activate the environment changes.

- For WebLogic:

  a Shut down the application server if it is running.

  b Modify *$BEA_HOME/repra/bin/repra_env.sh* to add the full path of *mycustom.jar* or the *customclasses* directory to the end of the CLASSPATH definition.

  Do *one* of the following:

  - Enter:

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

  - Or, enter:

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

  c Start the WebLogic Server to activate the environment changes.

Example output for TIBRV Listener and Active Enterprise wire format

TIBRV listener

For a TIBRV listener, the output is as follows.

Note The following message has 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]
}
}
```

Active Enterprise wire format

For an Active Enterprise wire-formatted message generator, the output is as follows.

Note The following message has been formatted for readability.

```
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:8080/repra/dtds/dbeventstream.dtd'>
      <dbStream environment="repraJMS2.repdb">
          <tran
              eventId="102:000000000000c8d500007fe7002a00007fe70028000093b
      d00bd716e0000000000010001">
             <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 }
      }}
```

APPENDIX A

Configuration Worksheets

This appendix contains worksheets on which you can record all 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 or messaging system you have. See Chapter 2, "Overview of RepConnector Configuration," for more information.

Table A-1: Replication Server information (Chapter 3)

	Current value	Example	Maps to
DSI info			
1) Used by Replication Server to identify where RepConnector connection will run. This information is 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	

	Current value	Example	Maps to
Replicated Database information needed by Replication Server			
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 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	33.n		

Table A-2: JMS System information

	Current value	Example	Maps to
Destination Type		queue	Outbound or inbound messaging – JMS information
JMS Provider URL		iiop://localhost:9000	
Initial Naming Context Factory		com.sybase.jms.IntialContextFactory	
Connection Factory		javax.jms.QueueConnectionFactory	
Destination Name		sampleQueue	
User Name		jagadmin	
Password			
Topic Subscribers			
Status Destination			

Table A-3: TIBCO RBV messaging system information

	Current value	Example	Maps to
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 A-4: IBM WebSphereMQ messaging system information

	Current value	Example	Maps to
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	
User Name		mquser	
Password		mqpass	
Status Definition			

Table A-5: Database system information

	Current value	Example	Maps to
JDBC Connection URL		jdbc:sybase:Tds:localhost: 5000	See worksheet for Chapter 3
Driver Class		com.sybase.jdbc2.jdbc. SybDriver	
User Name		sa	
Password			

Table A-6: Customization information (Chapter 6)

	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 A-7: RepConnector profile properties

	Current value	Example	Maps to
Profile Name		EAS:onXP	See worksheet for Chapter 3
Host Name		localhost	
Port Number		8080	
User Name		repraadmin	
Password			

Table A-8: RepConnector connection: General properties

	Current value	Example	Maps to
Inbound Type		Replication	Used by RepConnector
Outbound Type		JMS	
DBEventStream XSD URL		http://localhost:8080/ RepraWebApp/dtds/ dbeventstream.xsd	
Log Level		INFO	
Auto Start Connection		FALSE	
Customized Plug-in Class		FALSE	

Table A-9: Inbound RepConnector connection: DSI properties

	Current value	Example	Maps to
DSI Name		RepConnector	See worksheet for Chapter 3
DSI Port		7000	
User Name		sa	
Password			

Table A-10: Inbound RepConnector connection: RSSD properties

	Current value	Example	Maps to
RSSD URL		jdbc:sybase:Tds:userXP2:5000/rs/125XP_RSSD	See worksheet for Chapter 3
User Name		sa	
Password			
Grouping		FALSE	

APPENDIX B

Troubleshooting

This appendix discusses how to troubleshoot scenarios you may encounter in the RepConnector environment.

When the profile login or ratool fails

This section describes the steps you should take if you cannot log in to the RepConnector connection profile.

Verifying application server environment

Verify that the application server is up and running, and has successfully called *repra_env.sh*.

For EAServer

1 Use a text editor to open the *user_setenv.sh* file:

```
vi $JAGUAR/bin/user_setenv.sh
```

2 Verifyythat the *repra_env.sh* is called by looking for these lines:

```
if [ -f $JAGUAR/repra/bin/repra_env.sh ]
     then
     . $JAGUAR/repra/bin/repra_env.sh
fi
```

3 Add an echo statement to *repra_env.sh* to ensure that the RepConnector values are picked up when the application server is started. For example:

```
echo SERVER_TYPE : $SERVER_TYPE
```

For WebLogic Server

1 Using a text editor, open the *<domain_name>/startWebLogic.sh*:

```
vi $BEA_HOME/user_projects/domains/<domain_name>/startWebLogic.sh
```

2 Verify that *repra_env.sh* is called. For example, the following lines should appear above the server start line:

```
if [ -f /opt/bea/repra/bin/repra_env.sh ]
     then
     . /opt/bea/repra/bin/repra_env.sh
fi
```

3 Add an echo statement to *repra_env.sh* to ensure that the RepConnector values are read when the application server is started. For example:

```
echo "SERVER_TYPE : $SERVER_TYPE"
```

Verifying that the application server is called

Verify that *repra_env.sh* has the correct application server identified.

For EAServer

At the command line:

1 Using a text editor, open:

```
$JAGUAR/repra/bin/repra_env.sh
```

2 Verify that SERVER_TYPE is defined. If it is not, define the variable by entering:

```
SERVER_TYPE=EAS
```

For WebLogic Server

At the command line:

1 Open the following file with an editor:

```
$BEA_HOME/repra/bin/repra_env.sh
```

2 Verify that SERVER_TYPE is defined. If it is not, define the variable by entering:

```
SERVER_TYPE=WEBLOGIC
```

Configuring RepConnector for debugging

❖ **Setting the logging level to DEBUG for RepConnector runtime**

1 Navigate to the RepConnector runtime installation location's *bin* directory:

```
cd /opt/sybase/EAServer/repra/bin
```

2 In the *repra.properties* file, change the runtime log level from INFO to DEBUG:

```
RuntimeLogLevel=DEBUG
```

3 Shut down and restart the application server.

❖ **Setting the logging level to *debug* for each RepConnector connection**

1 Using RepConnector Manager, 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. You can use this information to troubleshoot failures, but be aware that a large *repra.log* file can cause performance degradation.

Verifying machine name and port number

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

- For EAServer, look at *$JAGUAR/bin/Jaguarhttpservlet.log*, where *$JAGUAR* points to the EAServer installation location. For example; `/opt/sybase/EAServer.`
- For WebLogic Server, look at *$BEA_HOME/user_projects/domains/<DomainName>/<DomainNameServer>/<DomainNameServer>.log*, where *$BEA_HOME* points to the WebLogic Server installation location. For example; `/opt/bea.`

Verifying user name and password

The default user name for RepConnector Manager is “repraadmin” with a blank password. If you change the default, run *setlogin.sh* before you attempt to log in to RepConnector Manager. See Chapter 4, “Getting Started with RepConnector Manager,” for more information.

When a connection fails

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. See “Configuring RepConnector for debugging” on page 114.

Example from repra.log

This example displays a typical log in detail mode.

```
[RepToEASJMS]: 09:35:32 [INFO] [REP.RepAdapterImpl]: Starting Connection
RepToEASJMS.
[RepToEASJMS]: 09:35:33 [INFO] [JMS.JMSQueueClient]: Starting the JMS Queue
Client.
[RepToEASJMS]: 09:35:33 [INFO] [JMS.JMSQueueClient]: JMS Client is
configured successfully to be able to send event(s) to queue: INQ for
provider iiop://cmercer-xp:9000.
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepAdapterImpl]: Successfully
established client connection.
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepMetaConnection]: Executing query:
>>>> select dbname from rs_databases where dsname = 'RC25XPEAS'
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepMetaConnection]: Got the new
instance of SybDriver
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepMetaConnection]: Got a RSSD
connection to URL: jdbc:sybase:Tds:cmercer-
pc2:5000/rs125pc_RSSD?SQLINITSTR=set quoted_identifier off Login: sa
[RepToEASJMS]: 09:35:33 [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 from rs_repdbs, rs_subscriptions, rs_objects where ((dsname
= 'RC25XPEAS' and dbname = 'EAS422') 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 from
rs_repdbs, rs_articles, rs_subscriptions, rs_objects where  ((dsname =
'RC25XPEAS' and dbname = 'EAS422') 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)
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepMetaConnection]: Executing query:
>>>> select * from rs_columns where objid = 0x010000650000007A order by
colnum
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.DataServer]: A new RepListener was
added to the RepEventStream.
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepEventStream]: Added a Listener
without RequiredGroup option
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.RepEventStream]: RepAdapterListener
requires parsing and meta-data formatting.
[RepToEASJMS]: 09:35:33 [DEBUG] [REP.DataServer]: A new listener is added
to the stream object.
[RepToEASJMS]: 09:35:34 [DEBUG] [REP.ReplicateDB]: Created the dsi (.ser)
file :
\Program Files\Sybase\EAServer\repra\sers\DSI_RC25XPEAS_EAS422.ser
[RepToEASJMS]: 09:35:34 [DEBUG] [REP.RepMetaConnection]: Executing query:
>>>> select dsname, dbname from rs_databases where dsname = 'RC25XPEAS' and
dbname = 'EAS422'
[RepToEASJMS]: 09:35:34 [DEBUG] [REP.DataServer]: Put the DB connection to
the Hashtable of ReplicationDBs.
[RepToEASJMS]: 09:35:34 [DEBUG] [REP.RepAdapterImpl]: DataServer is now
ready.
[RepToEASJMS]: 09:35:34 [DEBUG] [REP.DSIReceiver]: jTDS server for DSI
RC25XPEAS is ready to listen on port 7051.
```

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:8080/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 on the Inbound Information screen

Verify that these fields are set correctly:

- DSI Name – must be the exact name added to Replication Server the *interfaces* file and the connection name used when creating the connection in Replication Server.

- Port – can be any available port; however, it should be the port identified in the Replication Server in the *interfaces* file.
- User Name/Password – the same name identified when you created the connection in Replication Server.

See Chapter 3, "Configuring Replication Server for RepConnector" and Table B-2 on page 120.

Verifying Replication Server system database information

Verify:

- RSSD URL – should be in this format: `jdbc:Sybase:Tds:<ServerName>:port` for the location of the Replication Server.
- User Name/Password – the user name and password used in Replication Server for connectivity.

  Use the ping feature in RepConnector Manager to validate user name and password.

Verifying inbound/outbound messaging systems

This section identifies three common verification error messages and provides a solution for each.

Problem: *Repra.log* example entry: java.lang.ClassNotFoundException.

Workaround: Check the RepConnector environment file, *repra_env.sh*. The RepConnector environment files are located in the RepConnector runtime installation's *bin* directory. For example: `/opt/sybase/EAServer/repra/bin.`

Note 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 IBMMQ 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/outbound messaging systems.

Verifying database connection information

Verify:

- That the JDBC Connection URL is in this format:
  `jdbc:Sybase:Tds:<ServerName>:port`
- The user name and password.

Use the ping feature in RepConnector Manager to validate the status of the database connection information.

Troubleshooting the replication system

This section provides troubleshooting techniques for Adaptive Server Enterprise and Replication Server.

Sybase Adaptive Server (primary data base)

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

The error log file is *$SYBASE_ASE/install/error.log*, where *$SYBASE_ASE* points to the Adaptive Server Enterprise directory in the installation location.

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
```

Table B-1: Adaptive Server commands

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	Gives status of replicated tables
sp_setrepproc <proc_name>,function	Set proc for replication

Replication Server

Use the Replication Server log file to see a detailed problem description.

The log file is *$SYBASE/REP-12_6/install/<RepServerName>.log*, where *$SYBASE_REP* points to the Replication Server installation.

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
```

Table B-2: Replication Server commands

Command	Action
admin who	Show status of connections
admin who_is_down	Show status of connections that are down
admin who_is_up	Show status of connections that are up
resume connection to <connection_name>	Resume connection
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 DSI
trace “on”, SQT,SQT_TRACE_COMMANDS	Set trace for SQT

Using *admin who* for your connection

admin who shows the current status of connections. The output below 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       118RC25XPEAS.EAS422
SQM         Awaiting Message       118:ORC25XPEAS.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

Sometimes, for troubleshooting purposes, restart all of the Replication Server and RepConnector components: Replication Server, EAServer, and the RepConnector connection.

Then suspend and resume all connections. Such a restart often clears what may be preventing a successful connection.

Purging Replication Server queues

When messages get delayed in Replication Server, try purging the queue. See the *Replication Server Reference Manual* commands for purging Replication Server queues.

Freeing transaction log space

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

Verifying sent messages

This section describes how to 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

D

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

U

V

W

X