SYBASE®

Configuration and Users Guide

RepConnector™

15.0.2

DOCUMENT ID: DC20112-01-1502-02

LAST REVISED: July 2009

Contents

About This Book .... **vii**

CHAPTER 1 **Overview** **1**

Introduction 1
RepConnector architecture 2
RepConnector process flow 3
Routing database events from Replication Server to messaging systems 3
Routing events from messaging systems to database tables ... 4
Guaranteed delivery 4
Status and error reporting 5
RepConnector configuration overview 5

CHAPTER 2 **Configuring Replication Server for RepConnector** **7**

Updating the interfaces file 8
Adding a RepConnector entry to the interfaces file 9
Creating the connection to RepConnector 10
Creating the replication definition 12
Creating and verifying the subscription 15
Resuming the connection to RepConnector 17

CHAPTER 3 **Getting Started with RepConnector Manager** **19**

Starting RepConnector Manager 19
Managing connection profiles 20

CHAPTER 4 **Configuring RepConnector** **25**

Before you configure connections 25
Configuring the RepConnector 26
Configuring RepConnector for SonicMQ JMS messaging systems 27
Configuring RepConnector for TIBCO 28
Configuring RepConnector for IBM WebSphere MQ 31

Configuring RepConnector for your database........................ 33
Configuring the environment for a custom sender or formatter 34
Configuring the environment for Replication Server routing.... 35
Configuring the environment to send real carriage return and tab 35
Creating RepConnector connections 36
Configuring replication information for REPLICATION inbound types 41
Configuring JMS information 43
Configuring TIBCO information 46
Configuring IBM MQ Information.. 48
Configuring custom plug-in information.................................. 49
Configuring database connection information 50
Managing a connection 50

CHAPTER 5 **Using the ratool Utility 55**
ratool utility 56
-copy 58
-delete 58
-getLogInfo 59
-getProperty 59
-import 60
-list... 61
-ping 61
-refresh... 63
-refreshAll... 63
-rename.. 63
-start 64
-startAll 64
-status 64
-stop 66
-stopAll 66

CHAPTER 6 **Customizing the Message Generator for TIBCO AECM............. 67**
Configuring properties for RepConnector 67
Connection configuration... 68
Property file containing the Active Enterprise Connection/Customization (ae.props) 68
Using the base class APIs 69
Default TIBCO AECM message generator............................. 69
Customized TIBCO AECM message generator...................... 69
APIs for a customized, wire-format message generator 71
APIs retrieving information from the source event 72

Configuring and using the default wire-formatted message generator 72
Configuring and using the customized wire-formatted message generator 74

CHAPTER 7 Customizing the Sender and Formatter Processors 79
Customizing the sender processor 80
RepraClient interface 80
Sample implementation of the RepraClient interface 81
RepraCustomClient interface 83
RepraCustomProps interface 83
Sample implementation of the RepraCustomClient interface.. 83
Customizing the formatter processor 87
RepTransactionFormatter interface 88
Sample implementation of the RepTransactionFormatter interface 89
Creating new custom sender and custom formatter classes 89
Using the DBEventParserFactory 89
DBEventParser APIs 90
package com.sybase.connector.repra.util; setSource(Object obj) throws Exception 90
int size() 90
String getDSName() throws Exception 90
String setDBName() throws Exception 90
String getEventId() throws Exception 90
String getOperation(int elemAt) 91
String getSchemaName(int elemAt) throws Exception 91
String getStatement() throws Exception 91
String setStatement(int elemAt) throws Exception 92
String getOwner(int elemAt) throws Exception 92
Vector getData(int elemAt) throws Exception 92
Vector getKeys(int elemAt) throws Exception 93
String getFieldName(Hashtable field) throws Exception 94
int getFieldType(Hashtable field) throws Exception 94
Object getFieldValue(Hashtable field) throws Exception 95
String toXMLText(String dtdURL) throws Exception 97
Using the RaXMLBuilder utility 98
RaXMLBuilder() 98
createTranDocument() throws Exception 98
createEventDocument() throws Exception 98
addOperation() throws Exception 99
addValue() throws Exception 99
addInValue() throws Exception 99
addOutValue() throws Exception 100

addWhere() throws Exception 100
write() throws Exception 100
xmlDocByteArray() throws Exception 100
xmlDocString() throws Exception 101
cancelOperation() throws Exception 101
getErrorEventId() throws Exception 101
getErrorStatusCode() throws Exception 101
getErrorMessage() throws Exception 101
String getOwner(int elementAt) throws Exception 101
Configuring the RaXMLBuilder 102
Running a sample implementation 104
Handling error messages 106
Compiling and running the sample 106
Handling ownership information 107

APPENDIX A **Configuration Worksheets 109**

APPENDIX B **Troubleshooting 115**
When the profile login or ratool fails 115
Verifying application server environment 115
Verifying machine name and port number 116
Verifying user name and password 117
Configuring RepConnector for debugging 117
When a connection fails 118
Verifying connection information 119
Troubleshooting the replication system 121
Sybase Adaptive Server Enterprise (primary data base) 121
Replication Server 122
Using admin who for your connection 122
Restarting components and connections 123
Purging Replication Server queues 123
Freeing transaction log space 123
Verifying sent messages 124

Index 127

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, "Configuring Replication Server for RepConnector," describes how to configure Replication Server® to communicate with RepConnector.
- Chapter 3, "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 4, "Configuring RepConnector," describes how to configure a RepConnector environment.
- Chapter 5, "Using the ratool Utility," describes how to use ratool—the command line utility—to create, configure, and manage RepConnector connections.
- Chapter 6, "Customizing the Message Generator for TIBCO AECM," describes how to use the RepConnector API to create a customized message generator that works with RepConnector.
- Chapter 7, "Customizing the Sender and Formatter Processors," describes how to use the RepConnector API to create customized sender and formatter processors that work 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

See the following documents for more information:

- *RepConnector Installation Guide* – describes procedures for installing RepConnector.
- *RepConnector Release Bulletin* – contains last-minute information that was too late to be included in the books.

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

❖ **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 time frame 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.

  On Windows:

  ```
  set SYBASE = c:\sybase
  ```

  On UNIX:

  ```
  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:

  On UNIX: */work/sybase*

  On Windows: *\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.

Key	Definition
<>	Identifies variable where you replace the text with your own properties or values. Do not write the angle brackets.

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 15.0.2 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.

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

Internationalization support

To route a message event from a database to a messaging system, you must have a correctly encoded database event from Replication Server. There is no special setting for RepConnector, but make sure that the character set of Replication Server is the same, or compatible with, that of the primary database. For information on setting up Replication Server's character set, see the *Replication Server Design Guide*.

Routing a database event from a messaging system to an Adaptive Server® database requires that all characters in the message can be encoded into the character set of the destination Adaptive Server database. Again, there is no special setting for RepConnector. The message is translated into a SQL statement, and executed against the destination Adaptive Server database.

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

Topic	Page
Introduction	1
RepConnector architecture	2
RepConnector process flow	3
RepConnector configuration overview	5

Introduction

RepConnector provides a nonpolling, nontrigger based solution to database integration, building on the Replication Server noninvasive system to push database transactions into a traditional integration environment. With RepConnector, there is no need to poll a database for changes, or to add triggers for event notification.

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

RepConnector provides adapters for sending to TIBCO, IBM MQ, SonicMQ, or any J2EE-compliant JMS messaging system. It can also 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
- Can group database events into a single transaction

- Supports text and image datatypes
- Parses replication events and generates XML documents
- Transforms incoming database events into XML messages, and routes them into configured message queues
- Can transform incoming database events into their application-specific format
- Can route incoming database events to any destination
- Detects message events and routes them to database tables
- Supports EAServer 6.0.2 as well as WebLogic 10.0 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:

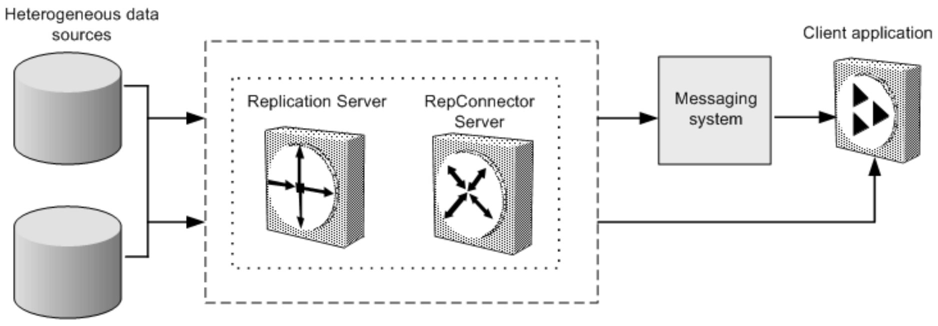
- Event Capture – listens for events from Replication Server or from the messaging system. Event Capture provides a TCP socket that listens for Replication Server events, and acts as a client for the messaging system, listening on the message bus for messaging events.
- Event Transformation – transforms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

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

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

Figure 1-1 provides a graphical representation of the RepConnector architecture.

Figure 1-1: RepConnector architecture



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 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 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 format, the Message Sender module applies the SQL statement to the database.

Guaranteed delivery

When routing replication events to messaging systems, RepConnector works with Replication Server to guarantee delivery. When RepConnector receives a message from Replication Server, it writes the message transaction ID to a file, ensuring that no transaction is lost in the event of a software failure. RepConnector then attempts to deliver the message. RepConnector does not send an acknowledgment to Replication Server until the message is successfully delivered to the specified messaging system. This process guarantees that no transactions are skipped, and that deliveries are sequential.

Status and error reporting

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

RepConnector configuration overview

RepConnector components include Sybase products as well as third-party products. You must configure each component before you can create RepConnector connections. The basic steps for configuring the RepConnector environment include:

1. Setting up Replication Server to send replicated events to RepConnector. See “Configuring Replication Server for RepConnector” on page 7.
2. Configuring your database server and messaging system in the RepConnector environment. See “Configuring RepConnector” on page 25.
3. Creating and configuring your RepConnector connection. See “Getting Started with RepConnector Manager” on page 19 and “Configuring RepConnector” on page 25.
4. Managing runtime control. See “Managing a connection” on page 50 for information about how your configuration of the RepConnector environment can influence your runtime control and management of RepConnector connections.

CHAPTER 2

Configuring Replication Server for RepConnector

Topics	Page
Updating the interfaces file	8
Creating the connection to RepConnector	10
Creating the replication definition	12
Creating and verifying the subscription	15
Resuming the connection to RepConnector	17

You or your system administrator must establish the RepConnector connection in Replication Server before you can configure 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 that contains 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* before you proceed.

To configure Replication Server to replicate to RepConnector:

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

As you work through 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. You must 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 connections DSI name, which you will need later when you configure the RepConnector connection.

  Note Sybase recommends that you use a name that clearly identifies this connection as allowing Replication Server to communicate with RepConnector, and that distinguishes it from a traditional connection between any data server and the corresponding database.

- Protocol – the network protocol for the DSI connection. This value should be recorded on line 3.b of the worksheet.

  You can use either the Transmission Control Protocol (TCP) or the NLWNSCK protocol on Windows, and either TCP or the Transport Layer Interface (TLI) TCP protocol on UNIX.

- Host name – the machine name where the RepConnector connection will be running. This value should be recorded on line 3.c of the worksheet.
- Port Number – the port where the RepConnector connection will be listening. This must be an unused port number on the host machine. Use the value recorded on line 3.d of the worksheet.

Adding a RepConnector entry to the interfaces file

In the interfaces file, on the machine on which Replication Server is running, create a new entry for the RepConnector connection

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

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

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

After you update the interfaces file, open the file to verify that your entry is correct. The Replication Server interfaces file is located in *%SYBASE%\ini\sql.ini* on Windows and *$SYBASE/interfaces* on UNIX, where *%SYBASE%* and *$SYBASE* are the locations of the Replication Server installation.

A sample entry for the interfaces file is:

- On Windows:

```
[server_name]
master=protocol, machine_name, port_number
query=protocol, machine_name, port_number
```

- On UNIX:

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

where:

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

Examples

This is an interface entry for the RepConnector connection:

- On Windows:

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

- On UNIX:

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

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

Creating the connection to RepConnector

A RepConnector connection defines the information that Replication Server uses to connect to the RepConnector server, to replicate data.

Before you create the connection, gather the following information:

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

❖ **Creating a new RepConnector connection in Replication Server**

1 Using isql, 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 connections DSI name. This is the same name you added to the Replication Server interfaces file. Use the name recorded in 3.a on your worksheet.
- *database* is the name of the database to which you are replicating. Record this value in 3.j on the worksheet.

Note When you create this connection in Replication Server, you must designate it with the *dataserver.database* data pair value. However, the database name in Replication Server is 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 and dsi_xact_group_size parameters as indicated above, use the configure connection and alter connection commands to set them. See the *Replication Server Reference Manual*.

For example:

```
create connection to RepConnector.RepCondb
set error class to rs_sqlserver_error_class
set function string class to
    rs_sqlserver_function_class
set username sa
set password null
set batch to 'off'
set dsi_xact_group_size to '-1'
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, skip this procedure.

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

❖ Creating a table replication definition in Replication Server

At the Replication Server, create the replication definition for the table you want to replicate. You must know the Adaptive Server name and database name in which the primary table resides.

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

❖ Creating a function replication definition

1. Gather the following information and record it on your worksheet, where appropriate:
   - Primary data server name (line 3.k)
   - Primary database name (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 the with replicate at command. 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.

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

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

For example:

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

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

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

where:

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

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

For example:

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

Resuming the connection to RepConnector

To ensure that Replication Server replicates commands to RepConnector, you must resume the connection from Replication Server to RepConnector.

❖ Resuming the connection

1. Gather:
   - 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.
   - *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
   ```

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

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

CHAPTER 3

Getting Started with RepConnector Manager

Topic	Page
Starting RepConnector Manager	19
Managing connection profiles	20

RepConnector Manager is a plug-in to the Eclipse framework that you can use to manage the activities of the RepConnector runtime components. You can run RepConnector Manager on the local machine where the runtime component is installed, or on any remote machine that can access the machine on which the runtime component is installed. To access a remote machine, verify that the HTTP connection to the application server on which the RepConnector runtime component is deployed and running, and is available throughout the network.

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

Starting RepConnector Manager

Start RepConnector Manager by running the *RepConnectorManager.bat* file on Windows or *RepConnectorManager.sh* file on UNIX.

❖ Starting Eclipse and RepConnector Manager

1 From a command prompt, navigate to the RepConnector Manager installation directory.

2 To start RepConnector Manager, enter:

- On Windows:

```
RepConnectorManager.bat
```

- On UNIX:

```
RepConnectorManager.sh
```

3 Select a workspace and click OK.

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

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

❖ Displaying the Sybase RepConnector Manager view

1 To display the Sybase RepConnector Manager view, perform one of these actions:

   - On the Welcome window, click the RepConnector Manager icon, or,
   - From the Eclipse menu bar:

     a Select Windows | Show View | Other.

     b In the Show View dialog box, select Sybase | RepConnector Manager.

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

Managing connection profiles

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 all the RepConnector installations from a single RepConnector Manager.

When you install RepConnector Manager, the installation provides two sample default profiles you can use to connect to your RepConnector runtime:

- EAServer: 8000 for Sybase EAServer
- WebLogic: 7001 for BEA WebLogic Server

❖ Creating a new profile

1 From the RepConnector Manager view, right-click the Sybase RepConnector folder, and select New Connection Profile to create a new RepConnector runtime profile.

2 Enter profile details:

- 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, colons, periods, hyphens, and underscores. It cannot contain white spaces.
- Host – the HTTP host name of the machine where the target RepConnector runtime is running. The default value is localhost.
- Port – the HTTP port number where the target RepConnector runtime is listening. The default is 8000 for EAServer and 7001 for BEA WebLogic.
- User – the RepConnector administrator user ID to connect to the RepConnector server. The default is repraadmin.

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

3 Select OK to create a new profile.

For example, enter:

```
Profile Name: EAServer:8000
Host: myhost
Port: 8000
User: repraadmin
```

❖ **Renaming a profile**

1 Right-click the profile to rename, and select Rename Profile.

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

3 Select OK.

❖ **Editing the profile properties**

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

2 Modify the fields and select OK.

❖ Deleting a profile

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

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 Server 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 must be equal to or less than 30 characters.

To modify the login:

1. From the command prompt, navigate to:

   On Windows:

   ```
   <EAServer_installation_folder>\repra\bin
   ```

   On UNIX:

   ```
   <EAServer_installation_folder>/repra/bin
   ```

2. Run this command to change the user name and password:

   On Windows:

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

   On UNIX:

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

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

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

❖ **Logging in to the RepConnector Server**

1 Start the RepConnector Server if it is not running.

2 Right-click the connection profile and select Login.

3 You see a window listing all of the RepConnector profile properties and an empty password field. Enter the password and click Login.

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

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

❖ **Refreshing the profile**

The tree view under the profile folder can be refreshed to show the current status of the runtime.

- Right-click the profile and select Refresh View.

❖ **Logging out of the profile and the RepConnector runtime**

Log out of the RepConnector profile to disconnect a profile.

1 Right-click the logged-in profile and select Logout.

2 Select OK.

CHAPTER 4 Configuring RepConnector

Topic	Page
Before you configure connections	25
Configuring the RepConnector	26
Creating RepConnector connections	36
Managing a connection	50

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 properly configured.

Before you configure connections

Before you begin to configure the connections, make sure you have started the:

- Application server and set up the messaging system.
- RepConnector Manager, created a connection profile, and connected to a RepConnector runtime instance. See “Getting Started with RepConnector Manager” on page 19.

You can create a RepConnector connection to:

- Listen for events from a database and 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 systems (JMS, TIBCO, IBMMQ) as the outbound type. The inbound type is the source from which the RepConnector Connection receives data. The outbound type is the destination to which the RepConnector connection routes the data.

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 systems (JMS, TIBCO, IBMMQ) as the inbound type and select DATABASE as the outbound type.

Configuring the RepConnector

Before validating the new RepConnector connection, you must configure your RepConnector environment, and restart your application server.

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

If you are configuring:

- A JMS Messaging System, go to “Configuring RepConnector for SonicMQ 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 31.
- For routing events from messaging to database, see “Configuring RepConnector for your database” on page 33.

- A customized plug-in, see "Configuring the environment for a custom sender or formatter" on page 34.

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

Configuring RepConnector for SonicMQ JMS messaging systems

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

If you do not configure your RepConnector environment before configuring the RepConnector connection, RepConnector Manager allows you to create the connection even if the ping has failed. However, you must go back to verify this connection once you have created the RepConnector environment.

❖ 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 the *repra_env.bat* file on Windows or the *repra_env.sh* file on UNIX that defines the SONICMQ_HOME environment variable is not commented out, and it points to the installation location for SonicMQ. For example:

   - On Windows:

     ```
     set SONICMQ_HOME=C:\SonicSoftware\SonicMQ
     ```

   - On UNIX:

     ```
     SONICMQ_HOME= /work/SonicSoftware/SonicMQ
     ```

2 Verify that the lines that define the directory structure for the SonicMQ library file, *sonic_Client.jar,* are *not* commented out and are correct for your environment.

On Windows:

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

On UNIX:

```
REPRA_CLASSPATH=$SONICMQ_HOME/lib/sonic_Client.jar
CLASSPATH=$CLASSPATH:$REPRA_CLASSPATH
BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
```

3 Restart your application server.

Configuring RepConnector for TIBCO

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 it is properly configured.

If you do not configure your RepConnector environment before configuring the RepConnector connection, RepConnector Manager allows you to create the connection even if the ping fails. However, once you have created the RepConnector environment, you must still verify the connection.

❖ Configuring TIBCO RV, RVCM

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

On Windows:

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

On UNIX:

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

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

On Windows:

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

On UNIX:

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

3 Restart your application server.

❖ Configuring TIBCO AECM

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

On Windows:

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

On UNIX:

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

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

On Windows:

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

On UNIX:

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

```
BOOTCLASSPATH=$BOOTCLASSPATH:$REPRA_CLASSPATH
```

3 Restart your application server.

❖ Configuring TIBCO Enterprise for JMS

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

On Windows:

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

On UNIX:

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

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

On Windows:

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

On UNIX:

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

3 Restart your application server.

Configuring RepConnector for IBM WebSphere MQ

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 correctly configured.

If you do not configure your RepConnector environment before configuring the RepConnector connection, RepConnector Manager allows you to create the connection even if the ping has failed. However, you must go back to verify this connection once you have created the RepConnector environment.

❖ Configuring IBM WebSphere MQ

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

   On Windows:

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

   On UNIX:

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

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

   On Windows:

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

   On UNIX:

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

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

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

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

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

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

4 Restart your application server.

❖ Configuring MQ JMS

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

On Windows:

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

On UNIX:

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

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

On Windows:

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

On UNIX:

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

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

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

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

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

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

4 Restart your application server.

Configuring RepConnector for your database

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

❖ Configuring for an Oracle database

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

On Windows:

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

On UNIX:

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

2 Restart your application server.

Configuring the environment for a custom sender or formatter

This section provides instructions for setting up the RepConnector environment so that RepConnector can load customized classes for message transformation or message destination routing.

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 detailed information about using the custom sender and formatter features.

❖ Configuring 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, in the EAServer server configuration property, the full path name to the *jar* file that contains the customized sender processor, the message formatter class, or the classes for the Java Classes property:

1. Log in to the EAServer Management Console.
2. On the General tab of the server property pages in the console, add the full path, along with your custom jar file, to the Java Class Path field. You can abbreviate the EAServer home directory by using "~", for example, "*~/repra/cust.jar*." See the *EAServer 6.0 System Administration Guide* for more information.
3. On the General tab of the Application.sybaserepconnector15, Class Loader page, add the full path, along with your custom jar file, to the Class Path field. You can abbreviate the EAServer home directory using "~"; for example, "*~/repra/cust.jar*."
4. Shut down EAServer and restart it.

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

1. Verify that the CLASSPATH variable setting in the *repra_env.bat* file on Windows, or the *repra_env.sh* file on UNIX, includes the full path of the *jar* file containing the customized sender processor or message formatter classes.
   - On Windows:

```
CLASSPATH=D:\bea\repra\sample\client\sample.jar;%CLASSPATH%
```

   - On UNIX:

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

2 Restart your WebLogic Server.

Configuring the environment for Replication Server routing

This section provides instructions for customizing the RepConnector environment to be used with Replication Server routing. Replication Server routing is when there is more than one Replication Server between the primary database and RepConnector.

❖ Setting up RepConnector for Replication Server routing

1 From the command prompt, navigate to:

On Windows:

```
<EAServer_installation_directory>\repra\bin
```

On UNIX:

```
<EAServer_installation_directory>/repra/bin
```

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

```
REMOTE_REPSERVER_OPTION=true
```

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

Configuring the environment to send real carriage return and tab

When there is a carriage return (\n) or tab (\t) in a source text column, RepConnector generates the literal representation "\n" or "\t" in the output stream. This section provides instructions on how to send the actual characters to the destination.

❖ Setting up RepConnector for real character output

1 From the command prompt, navigate to:

On Windows:

```
<EAServer_installation_directory>\repra\bin
```

On UNIX:

```
<EAServer_installation_directory>/repra/bin
```

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

```
REAL_CHARACTER_OPTION=true
```

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

Creating 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.

See Chapter 3, “Getting Started with RepConnector Manager,” to log in to the connection profile.

1 Right-click the 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 connection name. Do not use dashes or spaces.

   b Select the inbound type, which is the origin or source of the data.

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

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

- TIBCO – if this connection will accept inbound data from a TIBCO message queue. When you select TIBCO as the nbound type, you can select only DATABASE as the outbound type.
- IBM MQ – if this connection is to accept inbound data from an IBM WebSphere MQ message queue. When you select IBM MQ as the inbound type, you 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 name is "localhost."
- *<port_number>* is the port number of the target. The default port number is 8000.

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

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

Choose:

- FATAL – to see information about 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. When 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; 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 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 41.
- JMS – the JMS Information wizard page appears. See "Configuring JMS information" on page 43.
- TIBCO – the TIBCO Messaging Information wizard page appears. See "Configuring TIBCO information" on page 46.
- IBM MQ – the MQ Messaging Information wizard page appears. See "Configuring IBM MQ Information" on page 48.

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

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

Note You must update the *repra_env.bat* file on Windows or the *repra_env.sh* file on UNIX, to include the messaging system's libraries in the environment and restart your application server before you can use ping. See "Configuring the RepConnector" on page 26.

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:

- JMS – message queue or topic, the JMS Information page appears. See "Configuring JMS information" on page 43.
- TIBCO RV or RVCM message queue or topic – the TIBCO Messaging Information page appears. See "Configuring TIBCO information" on page 46.
- TIBCO AECM message queue – the TIBCO Messaging Information page appears. See "Configuring TIBCO information" on page 46.
- IBM MQ queues – the MQ Messaging Information page appears. See "Configuring IBM MQ Information" on page 48.
- CUSTOM – and selected Custom Plug-in Class in the General Connection Information wizard page, the Plug-in wizard page appears. See "Configuring custom plug-in information" on page 49.
- DATABASE – the Database Connection Information page appears. See "Configuring database connection information" on page 50.

11 Click Next.

The Summary page displays 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

If your inbound type is REPLICATION, you must enter the Replication Server inbound information and the Replication Server System Database information.

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.

You can find the values required for this procedure in Appendix A, “Configuration Worksheets,” 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 36.

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) to gather metadata information that RepConnector uses to process the events 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 36.

Configuring JMS information

If your connection is to a JMS message queue or topic, enter this information on the JMS Information window 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:2000`

  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:

     ```
     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. Enter or select 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 the 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 the destination type is a topic:

     ```
     weblogic.jms.TopicConnectionFactory
     ```

     Note 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 created in WebLogic Server.

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

     ```
     com.tibco.tibjms.
        TibjmsQueueConnectionFactory
     ```

If destination type is a topic:

```
com.tibco.tibjms.
   TibjmsTopicConnectionFactory
```

Note 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 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, enter:

```
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 36.

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.

See your TIBCO documentation for information about 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.
- 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 In Daemon, enter the TIBCO Rendezvous daemon value:

```
protocol:hostname:port
```

For example, enter `tcp:my_machine:7500`

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

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

To preregister listeners, the format on the "Subjects" line should be the main subject followed by name and subject pairs of the listeners that need to be preregistered. The main subject should be followed by a comma, then the name and subject pairs, separated by a colon. For example:

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

where, SAMPLE.REPC15.EVENT is the subject to publish to and cmNameA:cmSubA,cmNameB:cmSubB are the name and subject pairs that need to be preregistered.

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

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

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

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

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

3 If you selected AECM as the TIBCO message type, enter the 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 36.

Configuring IBM 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 an MQ message type:
   - MQ for IBM MQ Messaging System
   - MQJMS for IBM MQ JMS Messaging System
2. Choose:
   - Local Server, if you are running the MQ server daemon (IBM MQ server) on your local machine.
   - Local Client, if you are running the MQ client daemon (IBM MQ client) on your local machine.

   Select the encoding type 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 MQ server connection.

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

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

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

If you have just configured inbound information, return to step 10 in "Adding and configuring a new connection" on page 36.

If you have just configured outbound information, return to step 13 in "Adding and configuring a new connection" on page 36.

Configuring custom plug-in information

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

1 Custom Message Formatter – if you selected Customized Plug-in Class in the General Connection Information wizard, it means you will be loading a custom formatter. Enter the class name for your custom message formatter in the Message Formatter Plug-in Class field. For example, enter:

```
sample.MyMessageFormatter
```

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

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

2 Custom Sender – if you selected CUSTOM as your outbound type, it means you will be loading a custom sender. Enter the class name for your custom sender in the Sender Processor Plug-in Class field. For example, enter:

```
sample.FileSender
```

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

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

If you have just configured outbound information, return to step 13 in "Adding and configuring a new connection" on page 36.

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 33 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 where the data server is listening:

   ```
   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 36.

Managing a connection

This section assumes you have already started RepConnector Manager and have connected to the RepConnector instance. If you have not, see Chapter 3, “Getting Started with RepConnector Manager.”

Note You can also use the command line utility, ratool, to manage your RepConnector connections. See Chapter 5, “Using the ratool Utility” for more information.

❖ **Starting a connection**

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

❖ **Stopping a connection**

1 Right-click the connection and select Stop Connection.

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

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

❖ **Refreshing a connection**

1 Right-click the connection and select Refresh Connection.

2 When the connection has finished refreshing, click OK.

❖ **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.

3 Click OK.

❖ **Deleting a connection**

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

1 Right-click the connection and select Delete Connection.

2 Click OK. Confirm the deletion, or click Cancel to cancel the delete operation.

❖ **Copying a connection (Save As)**

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.

3 Click OK.

❖ **Validating a connection**

You can validate both the inbound and outbound configuration properties.

1 Right-click the connection and select Validate Connection.

2 Click OK.

❖ **Viewing or modifying properties for an existing connection**

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.
2. When you have finished viewing the log information, close the 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.
2. When you have finished viewing the runtime log information, close the 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 the profile and select View Log.

❖ **Refreshing the connection view display**

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

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

CHAPTER 5

Using the *ratool* Utility

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

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

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.

Topic	**Page**
ratool utility	56
-copy	58
-delete	58
-getLogInfo	59
-getProperty	59
-import	60
-list	61
-ping	61
-refresh	63
-refreshAll	63
-rename	63
-start	64
-startAll	64
-status	64
-stop	66
-stopAll	66

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 5-1 shows the valid options of ratool.

Table 5-1: ratool options

Options	Definitions
-host <*hostname*>	Identifies the name of the host on which the RepConnector runtime is running. The default is "localhost".
-port <*portnumber*>	The port number on which the RepConnector runtime instance is listening. The default value is 8000.
-user <*username*>	The RepConnector administrator login user ID. The default is 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 • INFO • DEBUG
Command options	
-copy <*conn_name*> <*new_conn_name*>	Copies a connection name to a new connection name.

Options	**Definitions**
-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 logging information for a specified connection. If you specify -file, this option writes the logging information to a file specified by *logFile*.
-import *<conn_name>* *<conn_prop_file>* [-override]	Adds a new connection. If -override is specified, 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 all connections.
-rename *<conn_name>* *<new_conn_name>*	Renames a connection name.
-start *<conn_name>*	Starts a specific connection.
-startAll	Starts all connections
-status *<conn_name>*	Lists connection status. If no connection name is specified, this command option gives the status of all known connections. Valid values are: • STARTING • RUNNING • STOP
-stop *<conn_name>*	Stops a specific connection.
-stopAll	Stops all running connections.
-validate *<conn_prop_file>*|| *<conn_name>*	Validates a new connection profile or an existing connection.

-copy

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.

If the destination connection already exists, this error message displays:

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

-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
If the connection is running, this option returns this message.

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

-getLogInfo

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 you specify -flag 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 d:\frepraconf\RepToJMS.props
```

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

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

Usage

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

-list

Description

Lists all known connections.

Syntax

ratool -list

Examples

To list all connections.

```
ratool -list
```

Usage

This option returns a list of known connections.

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

```
ratool -ping RepToJMS OUTBOUND
```

Example 4 To verify the replication configuration for a connection to RepToJMS by pinging Replication Server:

```
ratool -ping RepToJMS REPLICATION
```

Example 5 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** 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 8000, connecting as user “repraadmin” with no password:

```
ratool -status
```

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

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

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

Example 4 To send the debug logging information to a log file while running ratool, issue:

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

Example 5 If you have configured your application server to listen on a different port, for example, 8888, issue this command to display the status of all configured RepConnector connections:

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

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

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

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

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

Example 8 If you are connecting to RepConnector running on BEA WebLogic Servers default 7001 port, you can use:

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

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

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

-stop

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 6

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

Topic	Page
Configuring properties for RepConnector	67
Using the base class APIs	69

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. The user exit is where 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 6-1 lists the parameters for using the TIBCO Active Enterprise feature and the customized message generator:

Table 6-1: 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. For example: `AppConfig=C:/Sybase/EAS/repra/conf/ae.props`
AE Message Generator	The Customized Message Generator class name, and Active Enterprise-specific property. For example: `MsgGenerator=sample.MyMsgGenerator`

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

Table 6-2 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 6-2: 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/simple SDK_adapter`

Property name	Description
remote_repository	The location (full path) to the SDK repository. For example: `remote_repository=d:/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`
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 with no input argument. For example:

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

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

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

APIs for a customized, wire-format message generator

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 89 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 = d:\sybase\eas\conf\ae.props
AE Message Generator =
```

SDK application configuration with d:\sybase\eas\repra\conf\ae.props

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

pre_registered_subscribers=myCmListener,myCmListener2
```

Example output

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

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:8000/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 is a summarized overview of the different components from the back-end server, Adaptive Server® Enterprise, and Replication Server. It also 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=d:\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 on Windows or *$REPRA_HOME/sample/client/MyMsgGenerator.java* file on UNIX for an example for the customized AE message generator class.

❖ Compiling the customized message generator

1. Change to the location of your message generator:

   - On Windows:

     ```
     cd C:\work\custom
     ```

   - On UNIX:

     ```
     cd /work/custom
     ```

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

   - On Windows:

     ```
     md customclasses <enter>
     c:\jdk141\bin\javac -classpath .; C:\sybase\
     EAServer\repra\lib\repraconn.jar
     -d customclasses
     com\mycompany\MyMsgGenerator.java
     ```

   - On UNIX:

     ```
     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* on Windows or *com/mycompany* on UNIX.
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.
   - On Windows:

     ```
     cd C:\work\custom\customclasses
     ```

   - On UNIX:

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

2. Build the *jar* file:
   - On Windows:

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

   - On UNIX:

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

3. Add the path to *mycustom.jar* to your environment.
   - For EAServer:

     a. Copy the *jar* file or the directory structure containing the Java classes created from the previous step to the *java\classes* (Windows) or *java/classes* (UNIX) 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. From the pop-up, 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.bat* (Windows) or *$BEA_HOME/repra/bin/repra_env.sh* (UNIX) to add the full path of *mycustom.jar* or the *customclasses* directory to the end of the CLASSPATH definition.
    - On Windows, enter one of these:

      ```
      set
      CLASSPATH=C:\work\custom\customclasses;%CLAS
      SPATH%
      ```

      OR

      ```
      set
      CLASSPATH=C:\work\custom\customclasses\mycus
      tom.jar;%CLASSPATH%
      ```

    - On UNIX, enter *one* of these:

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

      OR

      ```
      CLASSPATH=/work/custom/customclasses/mycusto
      m.jar;$CLASSPATH
      ```

  - c Start the WebLogic Server to activate the environment changes.

Example output for TIBRV listener and Active Enterprise wire format

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]
            }
            }
```

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:8000/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 }
      }}
```

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

Topics	Page
Customizing the sender processor	80
RepraClient interface	80
RepraCustomClient interface	83
RepraCustomProps interface	83
Customizing the formatter processor	87
Creating new custom sender and custom formatter classes	89
Using the DBEventParserFactory	89
Using the RaXMLBuilder utility	98
Configuring the RaXMLBuilder	102

Note When you configure the RepConnector connection, you must indicate that you will be using a customized sender processor. See Chapter 4, “Configuring RepConnector.”

Customizing the sender processor

RepConnector has built-in senders that can post replication events to JMS, TIBCO, and IBM MQ messaging systems. It also provides an API for creating custom senders that can route replication events to anywhere that is accessible to Java, such as e-mail applications, files, or printers. To create a customized sender processor that runs within the RepConnector environment:

1. Create a class that implements either *com.sybase.connector.repra.RepraClient*, or *com.sybase.connector.repra.RepraCustomClient*. *RepraCustomClient* allows the RepConnector Manager to set and load a property page, but *RepraClient* does not.
2. Compile the class and archive it to a *.jar* file.
3. Using the RepConnector Manager, add a RepConnector connection that routes events to the custom sender processor:
   - Select Replication for the inbound type, and Custom for the outbound type.
   - On the Plug-in Class Information page of the wizard, enter the name of the custom class in the Sender Processor Plug-in Class. If *RepraCustomClient* is implemented, enter the path to and the name of the property page in Sender Processor Properties File.
4. Modify the application server environment to load the customized sender processor. See “Configuring the environment for a custom sender or formatter” in Chapter 4, “Configuring RepConnector.”
5. Shut down and restart the application server.

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)
    {
    }
}
```

RepraCustomClient interface

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

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

RepraCustomProps interface

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

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

Sample implementation of the RepraCustomClient interface

```
import java.io.FileInputStream;
import java.util.Date;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

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

/*
 * SampleClient2
 *
 * Description:This is a sample of a customer sender
 * processor client that will load a custom property
 * page and then route the event to an email address
 *
 */
public class SampleClient2 implements RepraCustomClient
{
    private Transport _transport = null;
    private MimeMessage _msg = null;
    private RaLogger _log = null;
    protected static String _host;
    protected String _from;
    protected String _to;
    protected String _cc;
    protected String _subject,

    protected String _username;
    protected String _password;
    private String _propFile;
    /**
    * sets the property file
    */
    public void setConfigProps(String custPropsFile)
{
        _propFile = custPropsFile;
}
    /**
    * gets the property file
    */
    public String getConfigProps()
    {
    return _propFile;
    }
    /**
        * sets the default logger
        */
        public void setLogger (RaLogger log)
    {
    if (_log == null)
        {
            _log = log;
        }
    }
    /**
 * gets the email information from the properties
 * @throws Exception
        */
        private void getHostInformation() throws
Exception
    {
        String thePropFile = getConfigProps();
        FileInputStream in =
newFileInputStream(thePropFile);
        _log.info ("SampleClient2.INFO", "** IN
getHostInformation, loading prop file");
        Properties prop = new Properties();
        prop.load(in);
        _host = prop.getProperty("MAIL_HOST","");
        _from = prop.getProperty("MAIL_FROM","");
        _to = prop.getProperty("MAIL_TO","");
        _cc = prop.getProperty("MAIL_CC","");
        _subject = prop.getProperty("MAIL_SUBJECT","");
        _username =
prop.getProperty("MAIL_USERNAME","");
        _password =
prop.getProperty("MAIL_PASSWORD","");
        _log.info ("SampleClient2.INFO", "** HOST - " +
_host + ", MAIL_TO - " +_to);
        }
        public void configureClient() throws Exception
        {
        try
        {
            _log.info ("SampleClient2.INFO", "** Starting
ConfigureClient");
            Properties prop =System.getProperties();
            Session ses  =
Session.getDefaultInstance(prop,null);
            getHostInformation();
            _msg = new MimeMessage(ses);
            _msg.setFrom(new InternetAddress(_from));
            _msg.addRecipient(Message.RecipientType.TO,
new InternetAddress(_to));
            _msg.addRecipient(Message.RecipientType.CC,
new InternetAddress(_cc));
            _msg.setSubject(_subject);
            _msg.setSentDate(new Date());
            _msg.saveChanges();
            _transport = ses.getTransport("smtp");
            _transport.connect(_host, _username,
_password);
            log.info ("SampleClient2.INFO", "** Ending
ConfigureClient");
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

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

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

    /**
        * returns true if the connection is healthy
     */
    public boolean isReady()
    {
        return _transport.isConnected();
    }
        /**
     * closes the client connection
     */
     public void close()
        {
    try
        {
            _transport.close();
        }
        catch (Exception ex)
        {
            // do nothing
        }
    }
}
```

Customizing the formatter processor

RepConnector translates replication events into XML before delivering them. It also provides a Java API that allows you to create custom formatters that translate replication events into other formats. To create a customized formatter that runs within the RepConnector environment:

1. Create a class that implements either *com.sybase.connector.repra.rep.RepTransactionFormatter* or *com.sybase.connector.repra.rep.RepraCustomTransactionFormatter*. *RepraCustomTransactionFormatter* allows the RepConnector Manager to set and load a property page, while *RepTransactionFormatter* does not.
2. Compile the class and archive it to a *.jar* file.
3. Using the RepConnector Manager, add or modify a RepConnector connection where the inbound type is "REPLICATION."
    - On the General Information page, select Customized Plug-in Class.
    - On the Plug-in Class Information page of the wizard, enter the name of the custom class in the Message Formatter Plug-in Class field. If *RepraCustomTransactionFormatter* was implemented, enter the path to the property page and its name in the Message Formatter Properties File field.
4. Modify the application server environment to load the customized message formatter. See "Configuring the environment for a custom sender or formatter" in Chapter 4, "Configuring RepConnector."
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 in the *<AppServer location>\repra\sample\client* directory on Windows, or *<AppServer location>/repra/sample/client* directory on UNIX, as a sample of the customized message formatter. The sample uses the DBEventParser utility to retrieve data and metadata from the replication event. See “Using the DBEventParserFactory” on page 89.

Creating new custom sender and custom formatter classes

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

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

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

Two samples, *CustomMessageFormatter.java*, and *MailClientCustom.java*, have been added to the RepConnector installation’s *sample/client* directory. MailClientCustom uses a custom configuration file named *sender.props*, which is 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.

Example

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

String getSchemaName(int elemAt) throws Exception

Description

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

Example

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

}
```

String getStatement() throws Exception

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 that represent the names, types, and values of the key field belonging to the operation at the position of elemAt. This method is meaningful only for update and delete operations.

The hash table 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
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

JDBC datatype constant	Constant value (int)
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:8000/RepraWebApp/dtds
    /dbeventStream.xsd));
```

Sample output:

```
    <!DOCTYPE dbStream SYSTEM
      'http://yjeongw2k:8000/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, addOperation()throws Exception returns null. Otherwise, it returns a String, the Element of the current operation.

Syntax

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

Parameters

operName – the 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 condition, 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 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 Windows, enter:

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

For UNIX, enter:

- For bsh:

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

- For csh:

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

❖ **Using the RaXML utility in your code**

1. Import the essential modules:

   ```
   import org.dom4j.Element;
   import com.sybase.connector.repra.util.*;
   ```

2. Create an instance of RaXMLBuilder:

   ```
   RaXMLBuilder raXML = new RaXMLBuilder();
   ```

3. Get the event body, which requires three parameters:

   - The URI of the *xsd.* file
   - The name of the database
   - The event ID of the current event, which can be any string value

   If you want the transaction (<tran>) type to contain multiple database operations, enter:

   ```
   foo.createTranDocument("file://dbeventstream.
      xsd","pubs2"","00001001");
   ```

   If you want the event (<dbevent>) type to contain a single database operation, enter:

   ```
   foo.createEventDocument("file://dbeventstream.
      xsd","pubs2"","00001001");
   ```

4 Add an operation, which requires:

- The command
- The name of the schema

For example:

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

5 Add data to the operation, which requires:

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

For example:

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

The field types, as SQL datatypes, are:

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

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

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

For example:

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

7 Create an XML file:

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

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

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

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

Running a sample implementation

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.:

- On Windows, enter:

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

- On UNIX, enter:

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

The result:

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


  dbStream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file://dbeventstream.xsd"
   environment="pubs2">
   <tran eventId="00001001">
   <update schema="authors">
    <values>
     <cell name="au_id" type="CHAR">0001</cell>
     <cell name="address" type="VARCHAR">1 Sybase</cell>
   <values>
   <oldValues>
    <cell name="au_id" type="CHAR" operator="=">0002</cell>
    <cell name="address" type="VARCHAR" condition="AND"
    operator="=">3 Sybase</cell>
   </oldValues>
  </update>
    <delete schema="authors">
     <values>
       <cell name="au_id" type="CHAR">0001</cell>
       <cell name="address" type="VARCHAR">1 Sybase</cell>
     </values>
   </oldValues>
     <cell name="au_id" type="CHAR" operator="=">0002</cell>
     <cell name="address" type="VARCHAR" condition="AND"
     operator="=">3 Sybase</cell>
     </oldValues>
    </delete>
    </update>
   </tran>
  </dbStream>

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

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


Output of a stored procedure:
```

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

Handling error messages

RepConnector sends any errors it encounters while processing an event to your Configure Status message queue. When you receive an error message from the queue, 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.

- On Windows, enter:

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

- On UNIX, enter:

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

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 this API 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:8000/RepraWebApp/dtds
           /dbeventstream.xsd" environment="RepConn.repdb">
           <tran
   eventId="102:000000000028cfc000007d8000f000007d8000c0000955700c0789e000
             000000001001">
           <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:8000/RepraWebApp/dtds
           /dbeventstream.xsd" environment="RepConn.repdb" owner="dbo">
           <tran
```

```
eventId="102:0000000000028cfc000007d8000f000007d8000c0000955700c0789e000
        000000001001">
        <delete schema=RepTable3">
         <oldValues>
          <cell name="repId" type="INT">1</cell>
         </oldValues>
        </delete>
        </tran>
      </dbStream>
```

APPENDIX A Configuration Worksheets

This appendix contains worksheets on which you can record all the configuration information for your RepConnector environment: Replication Server information, database information, application server information, messaging system information, and RepConnector information.

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

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

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

	Current value	Example	Maps to RepConnector Manager wizard
1) Used in create table for replication at database.			
2) Used in create replication definition for RepConnector at Replication Server.			
Primary DB to be replicated	3.j	pubs2	Not needed in the new Connection wizard
Host name of Database Server	3.k	primary_ase	
Port number where DB is listening	3.l	5000	
User name	3.m	sa	
Password	33.n		

Table A-2: JMS System information

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

Table A-4: IBM WebSphereMQ Messaging System information

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

Table A-5: Database system information

	Current Value	Example	Maps to
JDBC Connection URL		jdbc:sybase:Tds:localhost:5000	Outbound Type properties for an Outbound Type of database
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	RepConnector Manager, Connection Profiles
Host Name		localhost	
Port Number		8000	
User Name		repraadmin	
Password			

Table A-8: RepConnector connection: General properties

	Current value	Example	Maps to
Inbound Type		Replication	RepConnector Manager, General Properties
Outbound Type		JMS	
DBEventStream XSD URL		http://localhost:8000/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	RepConnector Manager, Inbound properties
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	RepConnector Manager, Inbound properties
User Name		sa	
Password			
Grouping		FALSE	

APPENDIX B Troubleshooting

Topic	Page
When the profile login or ratool fails	115
When a connection fails	118
Troubleshooting the replication system	121

When the profile login or ratool fails

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

Verifying application server environment

Verify that the application server is running, and has successfully called *repra_env.bat* on Windows, or *repra_env.sh* on UNIX. To do this, add an echo statement to *repra_env.bat* or *repra_env.sh*.

- On Windows:

```
echo SERVER_TYPE : %SERVER_TYPE%
```

- On UNIX:

```
echo SERVER_TYPE : $SERVER_TYPE
```

For WebLogic servers, verify that *repra_env.bat* or *repra_env.sh* is called correctly. *repra_env* must be called from the WebLogic startWebLogic command (startWebLogic.cmd on Windows, startWebLogic.sh on UNIX), in the *domain/bin* directory.

Example for UNIX

```
/work/software/bea10/user_prpojects/domains/mydomain/bin/startWebLogic.sh
```

The *repra_env* call must be made after WebLogic calls setDomainEnv (setDomainEnv.cmd on Windows, setDomainEnv.sh on UNIX).

Example *startWebLogic.sh* Before adding the repra_env call.

```
# Call setDomainEnv here
DOMAIN_HOME="/work1/software/bea10/user_projects/domains/mydomain"
.${DOMAIN_HOME}/bin/setDomainEnv.sh$*
SAVE_JAVA)OPTIONS="${JAVA_OPTIONS}"
SAVE_CLASSPATH="${CLASSPATH}"
```

Example *startWebLogic.sh* After adding repra.env call.

```
# Call setDomainEnv here.
DOMAIN_HOME="/work1/software/bea10/user_projects/domains/mydomain"
.${DOMAIN_HOME}/bin/setDomainEnv.sh$*
if [-f/work1/software/bea10/repra/bin/repra_env.sh]
then
./aseamd5_ work1/pherlich/software/bea10/repra/bin/repra_env.sh
fi

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

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

- On Windows – *%JAGUAR%\logs\<hostname>.log*, where %JAGUAR% points to the EAServer installation directory.
- On UNIX – *$JAGUAR/logs/<hostname>.log*, where $JAGUAR points to the EAServer installation directory.

For WebLogic Server

- On Windows – *%BEA_HOME%\user_projects\domains\<DomainName>\servers\<DomainNameServer>\logs\<DomainNameServer>.log*, where %BEA_HOME% points to the WebLogic Server installation directory.
- On UNIX – *$BEA_HOME/user_projects/domains/<DomainName>/servers/<DomainNameServer>/logs/<DomainNameServer>.log*, where *$BEA_HOME* points to the WebLogic Server installation directory.

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.bat* on Windows or *setlogin.sh* on UNIX, before you attempt to log in to RepConnector Manager. See Chapter 3, “Getting Started with RepConnector Manager” for more information.

Configuring RepConnector for debugging

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

1 From the command prompt, navigate to:

- On Windows:

```
cd c:\sybase\EAServer\repra\bin
```

- On UNIX:

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

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

```
LogLevel=DEBUG
RuntimeLogLevel=DEBUG
```

3 Shut down and restart the application server.

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

1 Use RepConnector Manager to log in to RepConnector runtime component.

2 Modify the connection properties to change the logging level (LogLevel) for the connection to DEBUG. Save the new connection properties.

3 Start or refresh the connection.

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

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 117.

This is a sample from a 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
: C:\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:8000/RepraWebApp/dtds/dbeventstream.xsd
```

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

Verifying Replication Server inbound information on Inbound Information screen

Verify that these fields are set correctly:

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

See Table A-1 on page 109.

Verifying Replication Server system database information

To verify Replication Server system database information, check for:

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

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

Verifying inbound or 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:

- On Windows: *RepConnector_install_dir\repra\bin\repra_env.bat*
- On UNIX: */opt/sybase/EAServer/repra/bin/repra_env.sh*

Note On UNIX, directory names are case sensitive.

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

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

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

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

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

Verifying database connection information

To verify database connection information, check for:

- JDBC Connection URL, which should be in this format: `jdbc:Sybase:Tds:<ServerName>:port`

- 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 Enterprise (primary data base)

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

- On Windows, the error log file is *%SYBASE_ASE%\install\error.log*, where *%SYBASE_ASE* points to the Adaptive Server installation directory.
- On UNIX, the error log file is *$SYBASE_ASE/install/error.log*, where *$SYBASE_ASE* points to the Adaptive Server installation directory.

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

Replication Server

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

- On Windows, the log file is *%SYBASE%\REP-15_2\install\<RepServerName>.log*, where *%SYBASE_REP%* points to the Replication Server installation directory.
- On UNIX, the log file is *$SYBASE/REP-15_2/install/<RepServerName>.log*, where *$SYBASE_REP* points to the Replication Server installation directory.

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 example 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        118 RC25XPEAS.EAS422
SQM         Awaiting Message        118: RC25XPEAS.EAS422
```

Changing connection grouping mode

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

Restarting components and connections

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. 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 this command:

   ```
   admin who,sqm
   ```

2. Check the output:

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

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

3. Determine the database ID and the queue type:

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

4. Put Replication Server into single-user mode:

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

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

6. Purge the queue:

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

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

7 Turn hibernate off:

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

Index

A

Adaptive Server Enterprise
  troubleshooting 121
addInValue function 99
addOoutValue function 100
addOperation function 99
addValue function 99
addWhere function 100
admin who, using 122
alter connection command 12
API
  DBEventParser 90
  RaXMLBuilder 98
application server
  verifying environment 115
application server support
  WebLogic 10.0 2
architecture
  Java Connector Architecture (JCA) 2

C

cancelOperation function 101
check subscription command 16
classes
  creating new 89
  developing 79
command options
  ratool 55
commands
  alter connection 12
  check subscription 16
  configure connection 12
  create connection 11
  create function replication definition 14
  create subscription 15
  DML 12
  resume connection to 17
  set dsi_xact_group_size 12
compiling
  RaXMLBuilder 106
components, restarting 123
configure connection command 12
configuring
  RaXMLBuilder 102
  Replication Server 7
  Replication Server to replicate to RepConnector 7
connection
  information, verifying 119
  resuming 17
  when fails 118
connection grouping mode, changing 123
connection name
  DSI 10
connection profiles
  managing 22
connections
  copying using **ratool** 58
  deleting using **ratool** 58
  displaying log information using **ratool** 59
  displaying status using **ratool** 64
  pinging using **ratool** 61
  renaming using **ratool** 63
  restarting using **ratool** 123
  stopping using **ratool** 66
conventions
  syntax x
-copy option 58
copying
  connections 58
create connection command 11
create function replication definition command 14
create subscription command 15
createEventDocument function 98
createTranDocument function 98
creating
  a replication definition in Replication Server 12
  a subscription 15

connection to RepConnector 10
function replication definition 14
replication definition 12
replication subscription 12
subscriptions 15
custom formatter
creating new classes 89
custom sender
creating new classes 89
customized formatter processor, creating 87
customizing sender and formatter processors 79

D

database 16, 17
database connection,verifying 120
database name variable 11
database tables
routing events from messaging systems to 4
database_name 15
dataserver 11
DBEventParser API 90
DBEventParserFactory utility 89
-delete
option 58
deleting
connections 58
RepConnector profiles 22
descriptions
creating a function replication definition 14
ratool 55
displaying
RepConnector profile 20
DML commands
RepConnector support 12
dsedit utility
using to add a RepConnector entry 9
DSI
connection name 10
dsiPassword 11
dsiUsername 11

E

editing
RepConnector profile properties 21
environment
verifying application server environment 115
error messages
RaXMLBuilder 106
Event Capture module 2
transforming messages to SQL format 4
Event Transfer module 2
Event Transformation module 4
examples
create connection to RepConnector 12
creating a subscription 16
RepraClient interface 80
set dsi_xact_group_size 12

F

features
RepConnector 1
formatter processor
customizing 79, 87
function
addInValue 99
addOperation 99
addOutValue 100
addValue 99
addWhere 100
cancelOperation 101
createEventDocument 98
createTranDocument 98
getData 92
getDSIName 90
getErrorEventId 101
getErrorMessage 101
getErrorStatusCode 101
getEventId 90
getFieldName 94
getFieldType 94
getFieldValue 95
getKeys 93
getOperation 91
getOwner 101
getSchemaName 91

  getStatement 91
  RaXMLBuilder 98
  setData 92
  setDBame 90
  setSource 90
  size 90
  toXMLText 97
  write 100
  xmlDocByteArray 100
  xmlDocByteString 101
function replication definition
  creating 14

G

getData function 92
getDSIName function 90
getErrorEventId function 101
getErrorMessage function 101
getErrorStatusCode function 101
getEventId function 90
getFieldName function 94
getFieldType function 94
getFieldValue function 95
getKeys function 93
-getLogInfo option 58, 59
getOperation function 91
getOwner function 101
-getProperty option 59
getSchemaName function 91
getStatement function 91
guaranteeing delivery 4

I

-import option 60
interface
  RepTransactionFormatter 88
interfaces file 7
  description 8
  information needed to update 8
  updating 8
  using dsedit utility 9
isql utility
  location 10

J

Java classes, developing for customizing 79

L

-list option 61
log information for connections
  display using **ratool** 59
login
  when fails 115

M

machine
  name, verifying logs 116
  port number, verifying logs 116
managing connection profiles 22
Message Sender module 2, 4
Message Transformation engine 2
messages
  sent, verifying 124
messaging systems
  routing events to database tables 4
messenger systems, routing events to 4
MSM (Message Sender module) 4

O

ownership information, RaXMLBuilder 107

P

parameters
  with replicate at 15
password, verifying 117
-ping option 61
pinging a connection using
  ratool 61

procedure, for customizing sender processor 80
profiles
  RepConnector 20

Q

queue
  Replication Server, purging 123

R

ratool 62
  command line utility 2
  description 55
  getting log information of connections 59
  option 58
  options 56
  ping connections 61
  renaming connections 63
  status of connections 64
  stopping connections 66
  syntax 56
  utility 55, 62
ratool command options
  -copy 56
  -delete 57
  -getLogInfo 57
  -getProperty 57
  -help 56
  -host 56
  -import 57
  -list 57
  -logfile 56
  -loglevel 56
  -password 56
  -ping 57
  -port 56
  -refresh 57
  -refreshAll 57
  -rename 57
  -start 57
  -startAll 57
  -status 57
  -stop 57
  -stopAll 57
  -user 56
  -validate 57
RaXMLBuilder
  compiling and running 106
  configuring 102
  function 98
  handling error messages 106
  ownership information 107
  sample implementation 104
  utility, API 98
-refresh option 63
-refreshAll option 63
-rename option 63
renaming connections
  using **ratool** 63
RepConnector
  architecture 2
  command line tool 55
  connection Name 15
  features 1
  profiles, editing properties 21
  profiles,deleting 22
  resuming connection 17
  resuming connections to Replication Server 17
  workflow 3
RepConnector Manager
  view, displaying 20
RepConnector_connection_name 17
replication definition
  creating 12
Replication Server
  configuring 7
  creating connection to RepConnector 10
  creating replication definition 12
  creating replication subscription 12
  inbound or outbound, verifying messaging systems information 120
  install worksheet 109
  purging queue 123
  system database information 120
  tasks for creating a replication definition in 12
  troubleshooting 122
replication system 121
  troubleshooting 121
replication system, troubleshooting 121

replication_definition_name 13, 15
RepraClient interface
  example 80
  sample implementation 81
RepTransaction Formatter
  interface 88
  sample implementation 89
restarting, components and connections 123
resume connection to command 17
resuming connections
  to RepConnector in Replication Server 17
routing events
  from messaging systems to database tables 4
  from Replication Server to messaging systems 3
routing replication events to messenger systems 4

S

sample implementation
  RaXMLBuilder 104
  RepraClient interface 81
  RepTransactionFormatter 89
sender processor, steps for customizing 80
sent messages, verifying 124
server support
  WebLogic 10.0 2
set dsi_xact_group_size 12
setData function 92
setDBName function 90
setSource function 90
size function 90
space, transaction log, freeing 123
SQL format
  Event Capture module 4
SQL to Event Transformation module 4
-start option 64
-startAll option 64
starting RepConnector Manager 19
status of connections
  display using **ratool** 64
-status option 64
-stop option 66
-stopAll option 66
stopping connections
  using **ratool** 66
subscription
  creating 15
  verifying 15
subscription_name 15
subscriptions
  creating 15
support
  DML commands 12
syntax conventions x

T

tasks
  configuring Replication Server to replicate to RepConnector 7
  creating a function replication definition 14
toXMLText function 97
transaction log space, freeing 123
transforming events
  into XML 3
transforming messages
  from XML to SQL format 4
troubleshooting 121
  Adaptive Server Enterprise 121
  Replication Server 122
  replication system 121

U

updating
  interfaces file 8
user name, verifying 117
utilities
  dsedit 9
  isql location 10
  ratool 2, 62

V

verifying
  subscription 15

W

WebLogic 10.0 Application Server 2
workflow, RepConnector 3
worksheets
- configuration 109
- database system information 109
- IBM WebSphere MQ 109
- JMS System 109
- Replication Server Installation 109
- TIBCO RBV 109

write function 100

X

XML
- transforming events into 3

XML to Message Sender module (MSM) 4
xmlDocByteArray function 100
xmlDocString function 101