SYBASE®

Administration Guide

Sybase Replication Agent™

12.5

DOCUMENT ID: 38261-01-1250-01

LAST REVISED: March 2002

Sybase, Inc., 5000 Hacienda Drive, Dublin, CA 94568.

Contents

About This Book

Sybase® Replication Agent™ version 12.5 extends the capabilities of Replication Server® by allowing non-Sybase (heterogeneous) database servers to act as primary data servers in a replication system based on Sybase replication technology.

Sybase Replication Agent is the Sybase solution for replicating table data changing operations and stored procedure invocations against a primary database in the following servers:

- IBM DB2 Universal Database (for UNIX and Microsoft Windows platforms)
- Informix Dynamic Server
- Microsoft SQL Server
- Oracle

This book describes Sybase Replication Agent administrative tasks and operations.

Audience

This book is intended for System Administrators and Database Administrators who are responsible for maintaining and monitoring primary databases, or a replication system based on Sybase replication technology (Replication Server).

How to use this book

The information in this book is organized as follows:

Chapter 1, “Introduction to Sybase Replication Agent,” provides an introduction to replication system concepts and an overview of the Sybase Replication Agent. This chapter describes the major Replication Agent components and explains how they work.

Chapter 2, “Setting Up Sybase Replication Agent,” describes the initial setup and configuration procedure for Sybase Replication Agent. The setup procedures described in this chapter must be performed after installing the software, and before replication can begin.

Chapter 3, “Administering Sybase Replication Agent,” describes administrative operations, including managing Replication Agent

instances and using the Replication Agent administration port to perform a variety of routine administrative tasks.

Chapter 4, “Sybase Replication Agent Command Reference,” describes all the Sybase Replication Agent commands in detail, including syntax, options, usage, and examples (where appropriate).

Chapter 5, “Sybase Replication Agent Configuration Parameters,” describes all the Sybase Replication Agent configuration parameters and the Replication Agent configuration file.

Chapter 6, “Troubleshooting Sybase Replication Agent,” describes basic troubleshooting and system recovery procedures for Sybase Replication Agent and the replication system.

Appendix A, “Administering the Replication Agent for UDB,” describes DB2 Universal Database-specific issues and details peculiar to the Sybase Replication Agent implementation for an IBM DB2 Universal Database primary database.

Appendix B, “Administering the Replication Agent for Informix,” describes Informix-specific issues and details peculiar to the Sybase Replication Agent implementation for an Informix primary database.

Appendix C, “Administering the Replication Agent for Microsoft SQL Server,” describes Microsoft SQL Server-specific issues and details peculiar to the Sybase Replication Agent implementation for a Microsoft SQL Server primary database.

Appendix D, “Administering the Replication Agent for Oracle,” describes Oracle-specific issues and details peculiar to the Sybase Replication Agent implementation for an Oracle primary database.

Appendix E, “Materializing Subscriptions to Primary Data,” describes the materialization process and provides a detailed procedure for materializing subscriptions to primary tables.

Related documents

A Sybase replication system comprises several components. You might find it useful to have the following documentation available as you install Sybase Replication Agent and set up your replication system.

Replication Agent

- The Sybase Replication Agent *Installation Guide* describes how to install and set up the Sybase Replication Agent software. It includes an installation and setup worksheet you can use to collect all the information you need to complete the Replication Agent installation.

- The *Replication Agent for Oracle Migration Guide* describes the migration process from Replication Agent for Oracle version 10.x or 11.x to Sybase Replication Agent.
- The Sybase Replication Agent 12.5 release bulletin contains current information that might not have been available when the Sybase Replication Agent guides were published.

Replication Server

- The Replication Server *Design Guide* introduces distributed database system concepts and describes how Replication Server supports distributed database systems, and how to design a replication system.
- The Replication Server *Administration Guide* contains information about Replication Server system administration issues.
- The Replication Server *Reference Manual* contains information about Replication Command Language (RCL) commands and syntax.
- The Replication Server *Heterogeneous Replication Guide* contains information about integrating heterogeneous (non-ASE and non-Sybase) database servers into a replication system based on Sybase replication technology.
- The Replication Server *Troubleshooting Guide* contains information to help diagnose and correct problems in a replication system.

Java environment

The Sybase Replication Agent 12.5 release bulletin contains information about Java requirements that may not have been available when the Sybase Replication Agent guides were published.

Java documentation available from your operating system vendor describes how to set up and manage your Java environment.

Further information about the Java environment on all platforms can be found at the following URL:

```
http://java.sun.com
```

Primary data servers

Sybase recommends that you or someone at your site be familiar with the software and database administration tasks for the primary database(s) supported by Sybase Replication Agent:

- IBM DB2 Universal Database user documentation describes how to install, configure, and administer an DB2 Universal Database data server. More information about DB2 Universal Database can be found at the IBM DB2 Web site:

  ```
  http://www.ibm.com/software/data/db2/
  ```

- Informix user documentation describes how to install, configure, and administer an Informix Dynamic Server data server. More information about Informix Dynamic Server can be found at the IBM Informix Web site:

  `http://www.ibm.com/software/data/informix/`

- Microsoft SQL Server user documentation describes how to install, configure, and administer a Microsoft SQL Server data server. More information about Microsoft SQL Server can be found at the Microsoft Web site:

  `http://microsoft.com/sql/techinfo/`

- Oracle database user documentation describes how to install, configure, and administer an Oracle database server. More information about Oracle database servers can be found at the Oracle Web site:

  `http://docs.oracle.com/`

Sybase Adaptive Server

Make sure you have the manuals appropriate for the version of Adaptive Server Enterprise you use with your replication system.

- The Adaptive Server Enterprise *Installation Guide* contains information about installing Adaptive Server Enterprise on your system. You can use this guide to help you install a data server for the Replication Server System Database (RSSD) or an Adaptive Server replicate database server.
- The Adaptive Server Enterprise *System Administration Guide* contains information you need to manage Adaptive Server.
- The Adaptive Server Enterprise *Utility Guide* describes isql and how to use it to query and manage database servers.

More information about Adaptive Server Enterprise can be found at the following URL:

`http://www.sybase.com/support/manuals/`

Other sources of information

Use the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- The Technical Library CD contains product manuals and is included with your software. The DynaText browser (downloadable from Product Manuals at http://www.sybase.com/detail/1,6904,1010663,00.html) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase/Powersoft newsgroups.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **For the latest information on EBFs and Updates**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select EBFs/Updates. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
3. Specify a time frame and click Go.
4. Select a product.
5. Click an EBF/Update title to display the report.

❖ **To create 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.

Conventions

The following sections describe content, style, and syntax conventions used in this guide.

Content conventions

Where possible, both UNIX and Windows (NT and 2000) operating system interfaces are shown. When it is not possible to show both UNIX and Windows interfaces, only the Windows interface is shown.

Style conventions

The following style conventions are used in this guide:

- Examples that show the use of Replication Server or Replication Agent commands appear like this:

  ```
  pdb_setreptable authors, mark
  ```

- Command names, command option names, parameter names, specific parameter values, program names, program command line options or arguments, and other keywords appear in body text like this:

  Use ra_set_login to set the administrative user ID.

- Syntax examples appear like this:

  alter user *username*
  set password *new_passwd*

  The words *username* and *new_passwd* in the syntax example are variables or user-supplied words.

- Variables and user-supplied words appear in body text like this:

  Set the new user password, *new_passwd*.

- Names of database objects (tables, columns, stored procedures, etc.) appear in body text like this:

  Check the base_price column in the Items table.

- Names of datatypes appear in body text like this:

  Use the date or datetime datatype.

- Examples of computer output appear like this:

  ```
  pub_id     pub_name           city          state
  -------     -------------------  -----------
  0736        New Age Books        Boston
  0877        Binnet & Hardley     Washington
  1389        Algodata Infosystems Berkeley

  (3 rows affected)
  ```

Syntax conventions

The following syntax conventions are used in this guide:

Table 1: Syntax statement conventions

Key	Definition
{ }	Curly braces indicate that you must choose at least one of the enclosed options. Do not include the braces when you enter the command.
[ ]	Brackets mean that choosing one or more of the enclosed options is optional. Do not include the brackets when you enter the command.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you may select only one of the options shown.
,	The comma means you may choose as many of the options as you like, separating your choices with commas to be typed as part of the command.

Syntax statements (showing the syntax and options for a command) appear like this:

```
ra_config param[, value]
```

Character case

Replication Agent command names are not case sensitive. In this book, most command syntax examples are shown in lowercase, however, you can disregard character case when typing Replication Agent command names. For example, typing PDB_XLOG, Pdb_Xlog, or pdb_xlog yields the same result.

Replication Agent configuration parameter names are case sensitive. For example, Scan_Sleep_Max is not the same as scan_sleep_max and the former would be interpreted as an invalid parameter name.

Database object names are not case sensitive in Replication Agent commands. If you need to use mixed-case object names in Replication Agent commands, you must delimit the object names with quote characters.

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

Introduction to Sybase Replication Agent

Sybase Replication Agent extends the capabilities of Replication Server by allowing non-Sybase (heterogeneous) database servers to act as primary data servers in a replication system based on Sybase replication technology.

This chapter provides an introduction to and overview of Sybase Replication Agent.

Basic replication system concepts

Transaction replication can be used to maintain data in separate databases called **replicate databases**. Replicate databases contain accurate, current copies or subsets of data from a **primary database**.

When a table in the primary database is marked for replication, transactions that change the data in that table are captured for replication. The primary database processes the transaction, and a copy of the transaction (including all its operations) is stored in the **transaction log**.

In the case of a stored procedure marked for replication, when the stored procedure is invoked in the primary database, all parameter values provided with the procedure invocation are captured and recorded in the transaction log. When a marked stored procedure generates a transaction that affects data in marked tables in the primary database, the transaction generated by the stored procedure is ignored, so only the procedure invocation is replicated.

Transaction replication

The events captured for replication through a Sybase replication system are referred to as *transactions*, even though they might not correspond directly to an actual transaction in the primary database. For example, if a transaction affects both marked tables and unmarked tables, only the operations that affect the marked tables are captured for replication. Operations on unmarked tables are ignored.

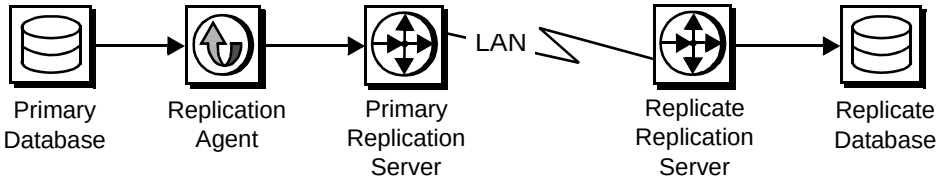
All data-change operations captured for replication exist within a *transaction context*. That is, only committed transaction operations are replicated; transactions that are rolled back are not replicated.

Even though the data-change events replicated through a Sybase replication system are really *operations*, those operations are grouped in an atomic collection, and they represent the results of a committed transaction in the primary database.

Replication system components

Figure 1-1 illustrates the basic components in a typical Sybase replication system.

Figure 1-1: Typical Sybase replication system



The following sections describe the primary-side components of a typical Sybase replication system:

- Primary databases
- Replication Agents
- Replication Servers

Primary databases

A primary database is the source of transactions that modify data in the replicate database(s). Transactions are replicated by table or by procedure.

Tables marked for replication in a primary database are called primary tables. A primary table must be marked for replication so that the Replication Agent can identify and replicate the transactions that affect the data in that table.

Large-object (LOB) columns within a primary table must have replication enabled separately from the primary table. You can selectively replicate LOB columns within a primary table.

To replicate invocations of a stored procedure, the procedure must be marked for replication so that the Replication Agent can identify and replicate invocations of that procedure in the primary database.

Replication Agents

A **Replication Agent** is the Sybase replication system component that captures the replicated transactions in a primary database, and then sends those transactions to a Replication Server for distribution to replicate databases.

Sybase Replication Agent reads a transaction log in the primary database and generates Log Transfer Language (LTL) output. LTL is the language that Replication Server uses to process and distribute replicated transactions throughout a replication system.

Sybase Replication Agent can be configured to use information stored in the Replication Server System Database (RSSD) of the primary Replication Server to provide more sophisticated replication features and generate more efficient LTL.

There are two types of Sybase Replication Agents:

- Trigger-based
- Log-based

Trigger-based Replication Agents

Trigger-based Replication Agents use *triggers* on marked tables to capture the data involved in a replicated transaction. The triggers also record other information the Replication Agent needs to replicate the transaction, such as a transaction ID that identifies each operation associated with a transaction.

Stored procedure replication is accomplished by similar triggers embedded in the stored procedure code.

When fired, the triggers record the data (or procedure invocation) to be replicated in one or more transaction log tables in the primary database. The transaction log tables are user tables created and maintained by the Replication Agent.

The Replication Agent creates the data-capture triggers in the primary database when a table or stored procedure is marked for replication. The triggers are removed by the Replication Agent when the table or procedure is unmarked.

Sybase Replication Agent uses the trigger-based solution for the following primary databases:

- Informix
- Microsoft SQL Server
- Oracle

Log-based Replication Agents

Log-based Replication Agents retrieve the information they need for transaction replication from the native transaction log maintained by the primary database server.

Sybase Replication Agent uses the log-based solution for primary databases in the DB2 Universal Database server.

Note Procedure replication is not available for DB2 Universal Database.

Replication Servers

The Replication Server that receives replicated transactions from a primary database (that is, directly from a Replication Agent) is referred to as the **primary Replication Server**. The Replication Server that sends replicated transactions to a replicate database is referred to as the **replicate Replication Server**.

Note In a simple replication system, a single Replication Server can act as both the primary Replication Server and the replicate Replication Server.

After it receives LTL from a Replication Agent, the primary Replication Server sends the replicated transaction to a replicate database, either directly or by way of a replicate Replication Server. The replicate Replication Server converts the replicated transaction from the LTL it receives to the native language of the replicate database, and then it sends the replicated transaction to the replicate database server for processing. When the replicated transaction is processed successfully by the replicate database, the replicate database is synchronized with the primary database.

Each Replication Server holds transaction operations in a stable queue and delivers them as soon as possible to other Replication Servers or replicate databases. By doing this, Replication Server guarantees transaction delivery; every transaction successfully received from a Replication Agent is guaranteed to be delivered to appropriately subscribing replicate databases.

Each Replication Server uses a database (called the RSSD) to store replication system data and metadata. Sybase Replication Agent can use some of the information stored in the RSSD to provide advanced replication features.

Understanding Sybase Replication Agent

Sybase Replication Agent supports transaction replication from a primary database through Replication Server. This section describes Sybase Replication Agent function in detail.

Note See the release bulletin for Sybase Replication Agent version 12.5 for information on the specific versions of DB2 Universal Database, Informix, Microsoft SQL Server, and Oracle supported by Sybase Replication Agent.

Sybase Replication Agent runs as a stand-alone application, independent of the primary database server, the primary Replication Server, and any other replication system components.

Sybase Replication Agent can reside on the same host machine as the primary database or any other replication system component, or it can reside on a machine separate from any other replication system components.

Sybase Replication Agent is compatible with Replication Server Manager (RSM). Replication Agent instances can be configured, managed, and monitored by RSM. In addition, you can completely configure, manage, and monitor a Replication Agent instance using any Open Client application that is capable of communicating with the Sybase Tabular Data Stream (TDS) protocol (such as isql).

Replication Agent instances

An instance of the Sybase Replication Agent (a Replication Agent instance) must be created for each primary database from which you want to replicate transactions. Each Replication Agent instance is an independent application with its own configuration and log files, administration port, and connections to the primary database and the primary Replication Server.

Replication Agent instances created for a specific primary database type are referred to in this book as follows:

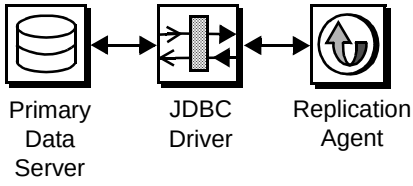
- IBM DB2 Universal Database – Replication Agent for UDB
- Informix Dynamic Server – Replication Agent for Informix
- Microsoft SQL Server (or SQL Server 2000) – Replication Agent for Microsoft SQL Server
- Oracle Server – Replication Agent for Oracle

Replication Agent communications

Sybase Replication Agent uses the Java Database Connectivity (JDBC) protocol for all communications. Some supported databases, however, require the Open Database Connectivity (ODBC) protocol. When connecting to a primary database, Sybase Replication Agent connects to either the JDBC driver or the JDBC/ODBC bridge provided by the database vendor.

Figure 1-2 illustrates the communication between Sybase Replication Agent and a primary database using a JDBC driver.

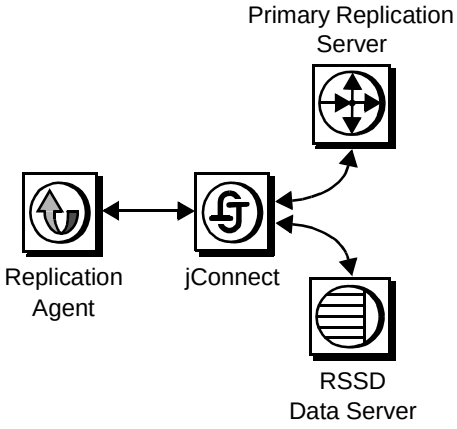
Figure 1-2: Replication Agent primary database communication



Sybase Replication Agent uses the Sybase JDBC driver (jConnect™ for JDBC™) to communicate with all Open Client/Open Server applications. Each Replication Agent instance uses a single instance of jConnect for JDBC.

Figure 1-3 illustrates the communication between Sybase Replication Agent and the primary Replication Server and its RSSD.

Figure 1-3: Replication Agent communication with Replication Server



While replicating transactions, the Replication Agent maintains connections with both the primary database and the primary Replication Server, and it may occasionally connect to the RSSD of the primary Replication Server to retrieve replication definition data.

Replication Agent components

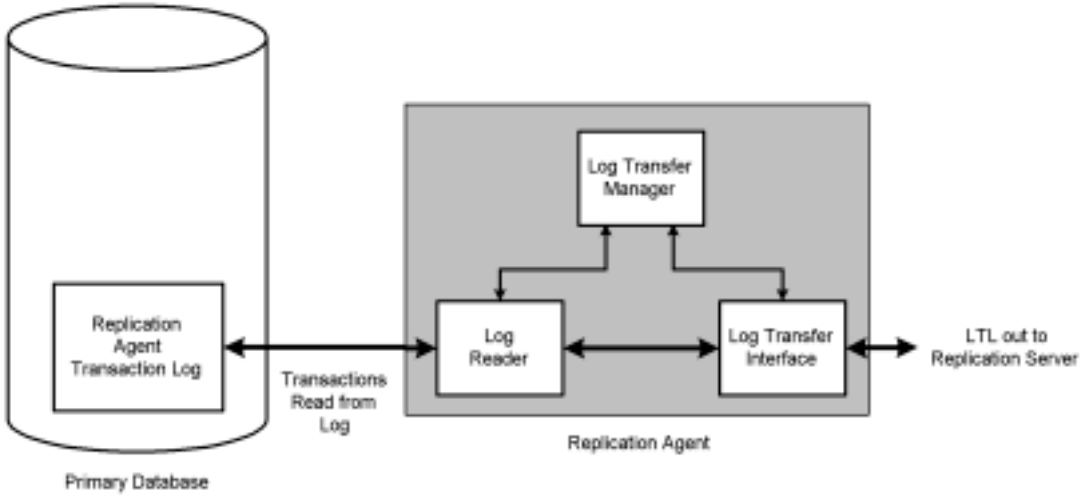
Sybase Replication Agent consists of a set of components that work together to perform all the operations required to propagate transactions from a primary database for replication.

The main Replication Agent components and their functions are:

- Log Reader – reads the transaction log in the primary database to retrieve transactions for replication.
- Log Transfer Interface (LTI) – generates Log Transfer Language (LTL) and sends it to the primary Replication Server.
- Log Administrator – administers the Replication Agent transaction log and manages transaction log objects.
- Log Transfer Manager (LTM) – manages all the other components and coordinates their operations and interactions.

Figure 1-4 illustrates the flow of data through a Replication Agent during replication.

Figure 1-4: Replication Agent operational data flow



The Log Reader component retrieves transaction data from the transaction log in the primary database, generates change set data, and passes the change sets to the LTI. The LTI component processes the change set data from the Log Reader and generates the LTL to send to the primary Replication Server. The LTI component also receives messages from the primary Replication Server.

Although the LTM component is not involved in the flow of data from the primary database to the primary Replication Server, it coordinates the activities of the other Replication Agent components and processes any errors generated by those components.

Administration port

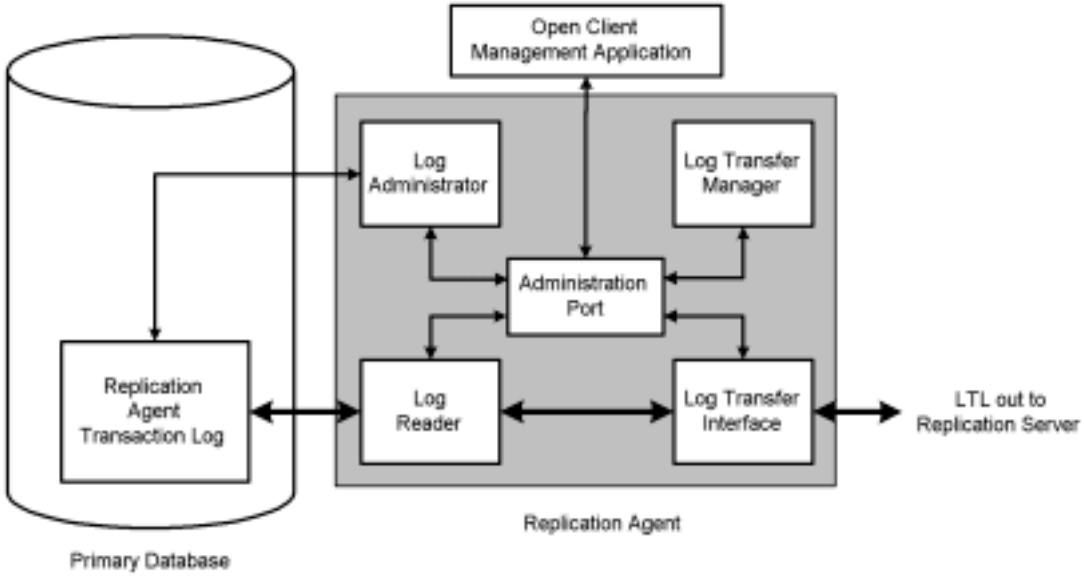
Sybase Replication Agent provides an administrative user interface through its *administration port*.

The administration port allows an Open Client application to log in to a Replication Agent instance, as if the Replication Agent were an Open Server application. Once logged in, the administrative client can issue commands to control, administer, and monitor the Replication Agent instance.

The administration port communicates with the client through the Sybase JDBC driver (jConnect for JDBC).

Figure 1-5 illustrates the flow of command and control messages among Replication Agent components.

Figure 1-5: Replication Agent command processing



The administration port passes commands from the administrative client to the Replication Agent components. The administration port also processes the messages from Replication Agent components, and passes those messages out to the client.

Java requirement

Sybase Replication Agent version 12.5 requires a Java Runtime Environment (JRE) on the computer that will act as the Replication Agent host machine.

For more information on installing and setting up a JRE, see:

- Sybase Replication Agent *Installation Guide*
- The release bulletin for Sybase Replication Agent version 12.5

CHAPTER 2

Setting Up Sybase Replication Agent

This chapter describes how to set up Sybase Replication Agent after the software is installed, verify that your replication system is ready to replicate, and start replication.

Note The procedures in this chapter assume you have already installed the Sybase Replication Agent software and established connectivity to the primary Replication Server and the primary database server, as described in the Sybase Replication Agent *Installation Guide.*

Preparing for replication

All the components of your replication system must be properly installed and configured before you can begin replication.

Use the following checklist to verify that your replication system is set up.

1 Create (or identify) the primary objects (tables or stored procedures) in the primary database.

2 Create (or identify) the replicate objects (tables or stored procedures) in the replicate database.

3 In the primary Replication Server (the Replication Server that controls the primary database), create a replication definition for each primary object (table or stored procedure) to be replicated.

4 In the replicate Replication Server (the Replication Server that controls the replicate database), create subscriptions for the replication definitions that were created in the primary Replication Server.

5 If the primary Replication Server is not the same as the replicate Replication Server, you must create routes between them.

6 Materialize each primary table in the replicate database so that the contents of the replicate database are synchronized with the primary database.

7 Create the Replication Agent transaction log objects in the primary database.

8 Mark and enable replication for the primary objects (tables or stored procedures) for which you want to replicate transactions from the primary database.

9 Start the Replication Agent instance and use the resume command to begin scanning the transaction log and sending LTL to the primary Replication Server.

Note Actual LTL output from the Replication Agent to the primary Replication Server does not occur until replicated transactions are captured in the primary database.

Sybase Replication Agent provides test scripts that automate these and other setup procedures for testing purposes (in a test environment). You can use the test scripts to verify the basic setup and configuration of your Sybase replication system.

For more information about Replication Agent test scripts, refer to the appropriate appendix in this book for your primary database server:

- Appendix A, “Administering the Replication Agent for UDB”
- Appendix B, “Administering the Replication Agent for Informix”
- Appendix C, “Administering the Replication Agent for Microsoft SQL Server”
- Appendix D, “Administering the Replication Agent for Oracle”

Starting a Replication Agent instance

There are two ways to start a Replication Agent instance:

- Using the ra command line utility
- Using the Administrator graphical user interface (GUI)

Start-up requirements

Before you can start a Replication Agent instance and connect to the primary database server, you must add the location of the JDBC driver for the primary database to the *CLASSPATH* environment variable.

See the Sybase Replication Agent *Installation Guide* for more information about installing and setting up the JDBC driver for the primary database and setting up Replication Agent connectivity.

Oracle requirements

If the primary database resides on an Oracle server, the TNS Listener Service must be installed and running on the primary database so the Replication Agent instance can connect to it.

See the Oracle networking guide for more information about TNS.

DB2 Universal Database requirements

If the primary database resides on a DB2 Universal Database server on a UNIX platform, you must set the UNIX environment variables before you start the Replication Agent instance.

See the Sybase Replication Agent *Installation Guide* for more information about setting the UNIX environment variables.

Using the ra command line utility

The Sybase Replication Agent ra command line utility allows you to start a Replication Agent instance. When you start a Replication Agent instance with the ra utility, you can specify the start-up state.

Note If you do not specify a start-up state on the command line when you invoke the ra command, the Replication Agent instance goes to its default *Admin* state.

❖ **To start a Replication Agent instance**

- Enter the following command at the operating system prompt:

  ```
  ra -iinst_name -state
  ```

  where *inst_name* is the name of the Replication Agent instance you want to launch and *state* is the keyword for the start-up state.

See "Using the ra utility" on page 29 for more information.

Note After you start a Replication Agent instance, you must open another operating system window to log in to the Replication Agent administration port and perform administrative tasks, such as shutting down, configuring, or changing the running state of the Replication Agent instance.

Using the Administrator GUI

❖ **To start a Replication Agent instance from the Administrator GUI**

1 If Administrator is not already running, start it by entering the following command at the operating system prompt:

```
administrator
```

Or, on Windows NT or Windows 2000, double-click the file name *administrator.bat* in File Manager or Windows Explorer. This file is located in the *rax-12_5\bin* subdirectory, under the directory in which you installed the Sybase Replication Agent software.

The Sybase Replication Agent dialog box opens.

2 Select the Replication Agent instance or instances you want to start.

Click Start.

The Confirm JDBC Driver dialog box prompts you to confirm that the appropriate JDBC driver is specified in the *CLASSPATH* environment variable.

Click Yes to continue.

Note If the appropriate JDBC driver is not listed in the *CLASSPATH* environment variable, you can start the Replication Agent instance, but the Replication Agent cannot connect to the primary database.

---

3 Click Start.

The Sybase Replication Agent dialog box indicates that the instance is running.

If the dialog box does not show that the instance is running, you may need to click Refresh to see if the Replication Agent instance is running.

On Windows NT and Windows 2000 platforms, each Replication Agent instance you selected is started in a console window using the ra_auto script, located in the *rax-12_5/bin* directory. Standard error output appears in the console window.

On UNIX platforms, each Replication Agent instance you selected is started in the background using the ra_auto script, located in the *rax-12_5/bin* directory. Standard error output is redirected to the *error.log* file, which is located in the Replication Agent instance's *log* directory.

Note If you start a Replication Agent instance with the same administration port number as an instance that is already running, Replication Agent aborts the process and records an error message in the system log.

4 Click Done to exit the Sybase Replication Agent dialog box.

See "Using the Administrator GUI" on page 38 for more information.

Note After you start a Replication Agent instance, you must open another operating system window to log in to the Replication Agent administration port and perform administrative tasks, such as shutting down, configuring, or changing the running state of the Replication Agent instance.

Setting up Replication Agent connectivity

Before you can create Replication Agent transaction log objects and start replicating transactions, you must set up connectivity between the Replication Agent instance and the following replication system components:

- Primary database
- Primary Replication Server
- RSSD of the primary Replication Server

Setting up connectivity for the Replication Agent requires:

- Setting values for Replication Agent configuration parameters
- Creating the primary database connection in the primary Replication Server
- Setting up the JDBC driver or JDBC/ODBC bridge for the primary database server
- Creating user logins for the Replication Agent with the appropriate authority in the primary database, primary Replication Server, and RSSD of the primary Replication Server.

For more information on setting Replication Agent parameters, creating the primary database connection in the primary Replication Server, and creating user logins in the primary Replication Server and RSSD, see the Sybase Replication Agent *Installation Guide*.

For more information on setting up the JDBC driver or JDBC/ODBC bridge for the primary database server and creating user logins in the primary database, see the appendix for your specific primary database type at the end of this Sybase Replication Agent *Administration Guide*.

Creating the Replication Agent transaction log

In trigger-based implementations (Informix, Microsoft SQL Server, and Oracle), Sybase Replication Agent uses its own proprietary transaction log to capture and record transactions in the primary database for replication. The Replication Agent transaction log consists of a set of database objects (tables, stored procedures, and triggers) that the Replication Agent creates in the primary database.

In the log-based implementation (DB2 Universal Database), Sybase Replication Agent uses the native transaction log maintained by the primary database to capture and record transactions. The Replication Agent for UDB also creates a few user tables in the primary database to support its operation.

Creating transaction log objects is the same process for both trigger-based and log-based Replication Agents.

Note All procedures in this book show isql used to log in to the administration port of the Replication Agent instance. You can use any other Open Client application to access the Replication Agent administration port.

❖ **To create a Replication Agent transaction log**

1 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user login name, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance you want to create a transaction log for.

Note If you are logging in to a newly created Replication Agent instance for the first time, use the default administrative user login sa and leave the password blank.

2 To define a prefix that uniquely identifies the Replication Agent transaction log you are creating, use the following command:

```
ra_config pdb_xlog_prefix, string
```

where *string* is a character string of one to three characters that will be used as a prefix for all database object names of the Replication Agent transaction log components created in the primary database.

Note The default value of the pdb_xlog_prefix parameter is ra_. Unless this string poses a conflict with existing database object names in your primary database, you can use the default value.

3 To create the transaction log, use the following command:

```
pdb_xlog create
```

When you invoke the pdb_xlog command with the create option, the Replication Agent generates a SQL script that is run in the primary database. This script creates the Replication Agent transaction log base objects.

Note Transaction log base objects must be created before any objects can be marked for replication in the primary database.

4 To verify that the Replication Agent transaction log was created, use the following command:

```
pdb_xlog
```

When you invoke the pdb_xlog command with no options, Replication Agent returns a list of the transaction log base objects in the primary database, if the transaction log was created successfully. If no information is returned, the transaction log does not exist in the primary database.

When the Replication Agent transaction log exists, and both primary database and Replication Server connections are defined correctly, you can put the Replication Agent instance in the *Replicating* state.

❖ **To put the Replication Agent instance in the *Replicating* state**

1 Log in to the Replication Agent administration port and use the following command at the isql prompt:

```
resume
```

Note The Replication Agent instance will go to the *Replicating* state only if a connection for the primary database has been created in the primary Replication Server. For more information on creating the primary database connection in Replication Server, see the Sybase Replication Agent *Installation Guide*.

2 To verify that the Replication Agent instance is in the *Replicating* state, use the following command at the isql prompt:

```
ra_status
```

The status information returned should indicate that the Replication Agent instance is in the *Replicating* state, which confirms that the transaction log exists and that all Replication Agent connections are up.

Marking objects in the primary database

There are three types of objects that can be marked for replication in a primary database.

- Tables
- Stored procedures
- Large-object (LOB) columns

Tables and stored procedures must be marked for replication and have replication enabled for the object (table or stored procedure). LOB columns must have replication enabled, and the table that contains the LOB column must be marked for replication and have replication enabled for that table.

Note Procedure replication is not available for DB2 Universal Database.

Marking tables in the primary database

For transactions against a table to be replicated, the primary table in the primary database must be marked for replication and replication must be enabled for that table.

❖ **To mark a table in the primary database**

1 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user login name, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance you want to mark a primary table for.

2 Use the pdb_setreptable command to determine if the table you want to mark is already marked in the primary database.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

- If the pdb_setreptable command returns information that the specified table is marked and replication is enabled, you need not continue this procedure.
- If the pdb_setreptable command returns information that the specified table is marked, but replication is disabled, skip step 3 and continue this procedure from step 4 to enable replication for the table.
- If the pdb_setreptable command returns information that the specified table is not marked, continue this procedure to mark the table for replication.

3 Use the pdb_setreptable command to mark the table for replication.

The pdb_setreptable command allows you to mark the primary table to be replicated and specify the name to use for replication.

- Use the following command to mark the table for replication using a replication definition with the same table name:

  ```
  pdb_setreptable pdb_table, mark
  ```

  where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

- Use the following command to mark the table for replication using a replication definition with a different table name:

  ```
  pdb_setreptable pdb_table, rep_table, mark
  ```

  where *pdb_table* is the name of the table in the primary database that you want to mark for replication and *rep_table* is the name of the table in the with all tables named rep_table clause in the replication definition for this table.

- When marking a table for replication, you can specify that the table owner's name is sent along with the table name in the LTL. To do this, use the owner keyword after the mark keyword, as shown in the following example:

  ```
  pdb_setreptable pdb_table, mark, owner
  ```

  where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

If the pdb_dflt_object_repl parameter is set to true, the table marked for replication with the pdb_setreptable command is ready for replication after you invoke the pdb_setreptable command successfully, and you can skip step 4 in this procedure.

Note The pdb_dflt_object_repl parameter is set to true by default.

If the pdb_dflt_object_repl parameter is set to false, you must enable replication for the table before replication can take place.

4 Use the pdb_setreptable command to enable replication for the marked table.

```
pdb_setreptable pdb_table, enable
```

where *pdb_table* is the name of the marked table in the primary database for which you want to enable replication.

After the table is marked and replication is enabled for the table, you can begin replicating transactions that affect data in that table.

Marking stored procedures in the primary database

To replicate invocations of a stored procedure in the primary database, the stored procedure must be marked for replication and replication must be enabled for that stored procedure.

Note Procedure replication is not available for DB2 Universal Database.

❖ **To mark a stored procedure in the primary database**

1 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user login name, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance you want to mark a stored procedure for.

2 Use the pdb_setrepproc command to determine if the stored procedure you want to mark is already marked in the primary database.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication.

- If the pdb_setrepproc command returns information that the specified stored procedure is marked and replication is enabled, you need not continue this procedure.
- If the pdb_setrepproc command returns information that the specified stored procedure is marked, but replication is disabled, skip step 3 and continue this procedure from step 4 to enable replication for the stored procedure.
- If the pdb_setrepproc command returns information that the specified stored procedure is not marked, continue this procedure to mark the stored procedure for replication.

3 Use the pdb_setrepproc command to mark the stored procedure for replication.

The pdb_setrepproc command allows you to mark the stored procedure to be replicated and specify the name to use for replication.

- Use the following command to mark the stored procedure for replication using a function replication definition with the same procedure name:

  ```
  pdb_setrepproc pdb_proc, mark
  ```

  where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication.

- Use the following command to mark the stored procedure for replication using a function replication definition with a different procedure name:

  ```
  pdb_setrepproc pdb_proc, rep_proc, mark
  ```

  where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication and *rep_proc* is the name of the stored procedure in the with all procedures named rep_proc clause in the function replication definition for this stored procedure.

If the pdb_dflt_object_repl parameter is set to true, the stored procedure marked for replication with the pdb_setrepproc command is ready for replication after you invoke the pdb_setrepproc command successfully, and you can skip step 4 in this procedure.

Note The pdb_dflt_object_repl parameter is set to true by default.

If the pdb_dflt_object_repl parameter is set to false, you must enable replication for the stored procedure so replication can take place.

4 Use the pdb_setrepproc command to enable replication for the marked stored procedure.

  ```
  pdb_setrepproc pdb_proc, enable
  ```

  where *pdb_proc* is the name of the marked stored procedure in the primary database for which you want to enable replication.

After the stored procedure is marked and replication is enabled for the stored procedure, you can begin replicating invocations of that stored procedure.

Enabling replication for LOB columns

For transactions that affect a LOB column to be replicated, the table that contains the LOB column must be marked for replication and have replication enabled, and the LOB column must have replication enabled.

If the value of the pdb_dflt_column_repl parameter is set to true, all LOB columns in a table have replication enabled automatically when you mark the table (by invoking the pdb_setreptable command). If the value of the pdb_dflt_column_repl parameter is set to false, you must enable replication separately for each LOB column before replication can take place.

Note The default value of the pdb_dflt_column_repl parameter is false.

❖ **To enable replication for a LOB column in the primary database**

1 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user login name, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance you want to mark a primary table for.

2 Use the pdb_setrepcol command to determine if replication is already enabled for the LOB column you want to enable replication for in the primary database.

```
pdb_setrepcol tablename, pdb_col
```

where *tablename* is the name of the table that contains the LOB column and *pdb_col* is the name of the LOB column in the primary database.

- If the pdb_setrepcol command returns information that replication is enabled for the specified column, you need not continue this procedure.
- If the pdb_setrepcol command returns information that replication is not enabled for the specified column, continue this procedure to enable replication for the column.

3 Use the pdb_setrepcol command to enable replication for the LOB column.

```
pdb_setrepcol tablename, pdb_col, enable
```

where *tablename* is the name of the table that contains the LOB column and *pdb_col* is the name of the LOB column in the primary database for which you want to enable replication.

After replication is enabled for the LOB column, you can begin replicating transactions that affect data in that column.

Starting replication

Note Before you attempt to replicate transactions, use the checklist in "Preparing for replication" on page 11 to verify that your entire replication system is set up properly.

❖ **To start replication in the Replication Agent instance**

1 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user login name, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance you want to start replication for.

2 Start replication by issuing the following command:

```
resume
```

3 Verify the Replication Agent instance is in the *Replicating* state by issuing the following command:

```
ra_status
```

When the Replication Agent instance is in the *Replicating* state, it is ready to scan the transaction log for transactions to be replicated and send LTL to the primary Replication Server.

If the Replication Agent instance is not in the *Replicating* state after you invoke the resume command, see Chapter 6, "Troubleshooting Sybase Replication Agent," for more information.

Using Replication Agent test scripts

Sybase Replication Agent provides a set of test scripts that help automate the process of creating a replication test environment that you can use to verify the installation and configuration of the Replication Agent software and the other replication system components.

See the appropriate appendix in this book for more information about the Replication Agent test scripts provided for your primary database server:

- Appendix A, “Administering the Replication Agent for UDB”
- Appendix B, “Administering the Replication Agent for Informix”
- Appendix C, “Administering the Replication Agent for Microsoft SQL Server”
- Appendix D, “Administering the Replication Agent for Oracle”

CHAPTER 3

Administering Sybase Replication Agent

This chapter describes administrative tasks and procedures for Sybase Replication Agent.

This chapter includes the following sections:

- Replication Agent administration overview
- Replication Agent utilities
- Replication Agent administration port
- Replication Agent administrative tasks

Replication Agent administration overview

There are only a few administrative tasks for Sybase Replication Agent. In general, once you install and set up a Replication Agent instance and mark the objects in the primary database that you want to replicate, Sybase Replication Agent requires little routine administration.

Most of the tasks described in this chapter are related to setting up a Replication Agent instance to operate within the replication system.

Replication Agent instances

A Replication Agent instance is a single, uniquely named Sybase Replication Agent installed to support a specific primary database. Each Replication Agent instance has its own configuration file, transaction log, and system log, and it manages its own connections to a primary database and to a primary Replication Server. Each Replication Agent instance has a subdirectory in the Replication Agent base directory.

When you install the Sybase Replication Agent software, you have the option to create Replication Agent instances as part of the installation process.

When you create a Replication Agent instance, you must specify:

- A unique instance name
- A unique client socket port number for its administration port
- The type of primary database the Replication Agent instance supports

You can create and run more than one Replication Agent instance on a single Replication Agent host machine, but each instance must have a unique name and a unique client socket port number for its administration port.

See the Sybase Replication Agent *Installation Guide* for more information on creating a Replication Agent instance when you install the software. See "Creating a Replication Agent instance" on page 34 in this guide for more information on creating a new Replication Agent instance after the software is installed.

Replication Agent utilities

There are two command line utility programs and a GUI utility provided with Sybase Replication Agent. These utilities facilitate creating and managing Replication Agent instances and starting up Replication Agent instances.

- ra – this command line utility starts a Replication Agent instance or returns the Sybase Replication Agent version number.
- ra_admin – this command line utility allows you to create, copy, verify, and delete a Replication Agent instance, or list all verifiable installed Replication Agent instances.
- administrator – this graphical user interface (GUI) utility allows you to create, manage, and start Replication Agent instances. This GUI utility requires a GUI interface on the Replication Agent host machine or on a workstation or terminal connected to the Replication Agent host.

Replication Agent utilities (ra, ra_admin, and administrator) are supplied as batch files for Windows NT and Windows 2000 and as shell scripts for UNIX operating systems. These files reside in the *bin* subdirectory in the Replication Agent base directory (*%SYBASE%\rax-12_5\bin*).

Using the ra utility

The Replication Agent ra utility provides two major capabilities:

- Starting a specified Replication Agent instance from an operating system prompt
- Determining the Replication Agent version number from an operating system prompt

To start the ra utility, invoke the ra command. The ra command has the following syntax:

ra [-help | -i *inst_name* [-*state*] | -v]

The options for the ra command are:

-help

returns usage information.

Note You can also invoke the ra command with no option specified to return command usage information.

-i *inst_name*

where *inst_name* is the name of an existing Replication Agent instance. If you invoke ra with the -i option specifying a valid Replication Agent instance name, that Replication Agent instance is started.

If you use the -i option, you can also use the -state option to specify the Replication Agent instance start-up state:

- -admin – Starts the Replication Agent instance in the *Admin* state. (This is the default start-up state.)
- -replicate – Starts the Replication Agent instance in the *Replicating* state.

-v

returns the Replication Agent version number.

Example

To start a Replication Agent instance named "NY2_RA" in the *Replicating* state, type the following command at the operating system prompt:

```
ra -i NY2_RA -replicate
```

For more information on Replication Agent states, see "Replication Agent states" on page 47.

Start-up errors

If errors occur while the Replication Agent instance is starting up, they are displayed on the command line or recorded in the Replication Agent system log file (*system.log*) on UNIX, or displayed on the command line on Windows NT or Windows 2000.

Using the ra_admin utility

The Replication Agent ra_admin utility provides the following major functions:

- Create, copy, delete, and verify a Replication Agent instance
- List all valid Replication Agent instances installed on the Replication Agent host machine
- Return the complete path of the Replication Agent base directory

To start the ra_admin utility, invoke the ra_admin command. The ra_admin command has the following syntax:

```
ra_admin [option [create options]] [inst_name]
```

Note You can also invoke the ra_admin command with no option specified to return command usage information.

The options for the ra_admin command are:

-b

returns the complete path of the Replication Agent base directory.

-c *inst_name*

where *inst_name* is a valid name, unique on the host machine, for a new Replication Agent instance. Creates a new Replication Agent instance using the specified name.

When the -c option is used, you must also specify the following create options:

-p *port_num*

where *port_num* is a valid client socket port number, unique on the host machine, for the administration port of the new Replication Agent instance.

-t *database*

where *database* identifies the type of database server that the primary database resides in. Valid options are:

- ibmudb – DB2 Universal Database
- informix – Informix Dynamic Server
- mssql – Microsoft SQL Server (valid only on Windows NT or Windows 2000)
- oracle – Oracle Server

Note The *database* option is not case-sensitive.

When the -c option is used, you also have the option of specifying that the configuration of the new Replication Agent instance should be based on the configuration file for an existing Replication Agent instance. To do this, use the -f option, as shown below:

-f *old_inst*

where *old_inst* is the name of an existing Replication Agent instance whose configuration you want to duplicate for the new Replication Agent instance.

When you use the -f option to copy an existing Replication Agent configuration, you need not specify the -t option. The database type is automatically copied when you specify the -f option.

Note When you invoke the ra_admin command with the -c option and specify the -f option, some of the configuration parameters are set to default values. See "Copying a Replication Agent configuration" on page 33 for more information.

-d *inst_name*

where *inst_name* is the name of an existing Replication Agent instance. Deletes the specified Replication Agent instance.

-l

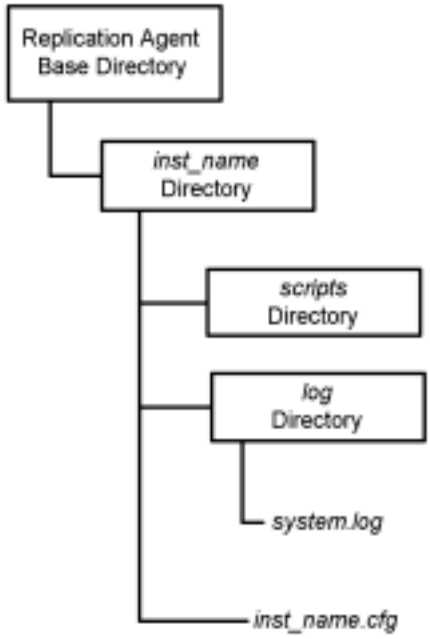
lists all verifiable Replication Agent instances.

-v *inst_name*

where *inst_name* is the name of an existing Replication Agent instance. Verifies the complete installed directory structure for the specified Replication Agent instance.

Figure 3-1 shows the directory structure created for each Replication Agent instance.

Figure 3-1: Replication Agent installed directory structure



The Replication Agent base directory (*rax-12_5*) is created when you install Sybase Replication Agent software on a host machine.

Note You can install the Replication Agent software more than once and create multiple independent Replication Agent base directories, however, a single installation (with a single base directory) can support multiple Replication Agent instances.

Each time you create a Replication Agent instance by invoking the ra_admin command with the -c option, the ra_admin utility creates the subdirectories for the Replication Agent instance under the Replication Agent base directory.

When you verify a Replication Agent instance by invoking the ra_admin command with the -v option, the ra_admin utility verifies the specified Replication Agent instance by checking for the subdirectory with the specified instance name.

When you delete a Replication Agent instance by invoking the ra_admin command with the -d option, the ra_admin utility deletes from the Replication Agent base directory all the subdirectories associated with the specified instance.

Note On Windows NT and Windows 2000, if any application or window is accessing a file or subdirectory associated with a Replication Agent instance at the time you delete the instance, the "open" file or subdirectory will not be deleted, and the deletion will be incomplete. An error message informs you of the file or subdirectory not deleted. In this event, you must verify that the file or subdirectory is not open in any application or window, and then manually delete it.

Copying a Replication Agent configuration

When you invoke the ra_admin command with the -c and -f options (to create a new Replication Agent instance and copy the configuration of an existing Replication Agent instance), the values of some configuration parameters in the existing configuration are not copied to the new configuration.

The values of the following configuration parameters are not copied from an existing configuration:

admin_port
log_directory
pds_database_name
pds_datasource_name
pds_host_name
pds_password
pds_port_number
pds_retry_count
pds_retry_timeout
pds_server_name
pds_service_name
pds_username
rs_source_db
rs_source_ds

See Chapter 5, “Sybase Replication Agent Configuration Parameters” for detailed information about these configuration parameters.

Creating a Replication Agent instance

You can create a new Replication Agent instance at any time after the Replication Agent software is installed by invoking the ra_admin command with the -c option.

The complete syntax of the ra_admin command with the -c option is:

ra_admin -c *new_inst* -p *port* {-t *database*|-f *old_inst*} -s

where *new_inst* is the name of the new Replication Agent instance you are creating, *port* is the client socket port number for the administration port of the new Replication Agent instance, *database* is the type of database server that contains the primary database, and *old_inst* is the name of an existing Replication Agent instance whose configuration you want to duplicate for the new Replication Agent instance.

You must specify either the -t option to identify the type of the database server or the -f option to copy an existing configuration. If you do not specify the -f option, the new Replication Agent instance is created with a default configuration.

When a Replication Agent instance is created, the configuration file for that instance is created automatically. The configuration file is named *inst_name.cfg*, where *inst_name* is the name of the Replication Agent instance. The configuration file resides in the *%SYBASE%\rax-12_5\inst_name* directory, where *%SYBASE%* is the Replication Agent installation directory.

Use the following procedure to create a new Replication Agent instance.

Note See the *Sybase Replication Agent Installation Guide* for information on how to create a Replication Agent instance as part of the installation process.

❖ **To create a Replication Agent instance**

1 Open a window with an operating system prompt.

2 Use the following command to make the Replication Agent *bin* directory the current directory:

```
cd \%SYBASE%\rax-12_5\bin
```

where *%SYBASE%* is the Replication Agent installation directory.

3 At the operating system prompt, invoke the ra_admin utility to create a new Replication Agent instance.

- If you want to create a new Replication Agent instance with a default configuration, issue the following command:

  ```
  ra_admin -c new_inst -p port -t database
  ```

  where *new_inst* is the name of the new Replication Agent instance, *port* is the client socket port number for the administration port of the new Replication Agent instance, and *database* is the type of database server in which the primary database resides.

- If you want to create a new Replication Agent instance whose configuration is based on the configuration of an existing Replication Agent instance, issue the following command:

  ```
  ra_admin -c new_inst -p port -f old_inst
  ```

  where *new_inst* is the name of the new Replication Agent instance, *port* is the client socket port number for the administration port of the new Replication Agent instance, and *old_inst* is the name of the existing Replication Agent instance whose configuration will be duplicated for the new instance.

After you invoke the ra_admin utility, the operating system prompt returns when the new Replication Agent instance is created.

4 Verify that the new Replication Agent instance was created properly using one of the following methods:

- Invoke the ra_admin utility with the -v option specifying the name of the new Replication Agent instance:

  ```
  ra_admin -v new_inst
  ```

  where *new_inst* is the name of the newly created Replication Agent instance.

- Invoke the ra_admin utility with the -l option:

  ```
  ra_admin -l
  ```

  The -l option lists all verifiable Replication Agent instances, which should include the new one you just created.

- As an alternative to using the ra_admin utility, you can use operating system commands to verify the existence of the correct Replication Agent instance directory structure (shown in Figure 3-1).

After you create a new Replication Agent instance, you can use the ra utility or the Administrator GUI to start the instance so it can be administered and configured.

Deleting a Replication Agent instance

You can delete a Replication Agent instance at any time by invoking the ra_admin utility with the -d option.

Before you delete a Replication Agent instance, you should:

- Remove the transaction log associated with that instance (if any). See "Removing the Replication Agent transaction log" on page 52 for more information.
- Shut down the Replication Agent application. See "Shutting down a Replication Agent instance" on page 45 for more information.
- If the Replication Agent software is installed on a Windows NT or Windows 2000 host machine, make sure that none of the files in the instance subdirectories are open and that no application or window is in any of the instance subdirectories.

Note On Windows NT and Windows 2000, if any application or window is in a file or subdirectory associated with a Replication Agent instance at the time you delete the instance, the "open" file or subdirectory will not be deleted and the deletion will be incomplete. An error message informs you of the file or subdirectory not deleted. In this event, you must verify that the file or subdirectory is not open in any application or window, and then manually delete it.

❖ **To delete a Replication Agent instance**

1 Open a window with an operating system prompt.

2 Use the following command to make the Replication Agent *bin* directory the current directory:

```
cd \%SYBASE%\rax-12_5\bin
```

where *%SYBASE%* is the Replication Agent installation directory.

3 At the operating system prompt, invoke the ra_admin utility with the -d option to delete a Replication Agent instance:

```
ra_admin -d inst_name
```

where *inst_name* is the name of the Replication Agent instance you want to delete.

After you invoke the ra_admin utility with the -d option, the following message appears:

```
Are you sure you want to delete the Replication Agent
instance inst_name? [y/n]
```

4 Enter y to delete the Replication Agent instance.

After the instance is deleted, the operating system prompt returns.

If the instance is running, you will receive an error message:

```
Cannot delete Replication Agent instance 'inst_name'
because it is currently running.
```

Note To delete an instance that is running, shutdown the instance first by logging into its administrative port and using the shutdown command, then delete it. See "Shutting down a Replication Agent instance" on page 45 for more information.

5 Verify that the Replication Agent instance was deleted.

There are several different ways to verify that a Replication Agent instance was deleted properly.

- Invoke the ra_admin utility with the -v option specifying the name of the deleted Replication Agent instance:

  ```
  ra_admin -v inst_name
  ```

  where *inst_name* is the name of the deleted Replication Agent instance.

- Invoke the ra_admin utility with the -l option:

  ```
  ra_admin -l
  ```

  The -l option lists all verifiable Replication Agent instances, which should not include the one you just deleted.

- As an alternative to using the ra_admin utility, you can use operating system commands to verify that the Replication Agent instance directory structure (shown in Figure 3-1) was removed.

Using the Administrator GUI

The Replication Agent Administrator GUI utility provides a convenient graphical user interface for the following major functions:

- Create, copy, delete, and start a Replication Agent instance
- List all valid Replication Agent instances installed on the Replication Agent host machine and view their current status

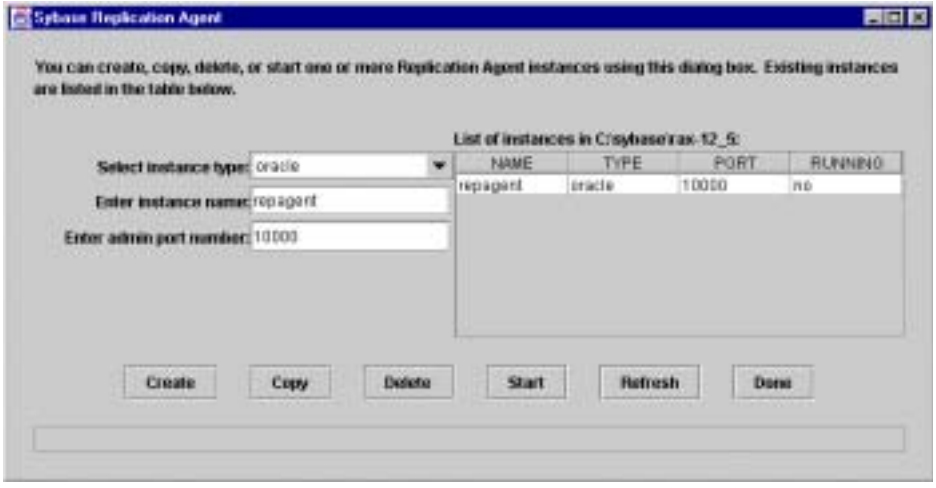
❖ **To start the Administrator GUI utility**

- Enter the following command at the operating system prompt:

  ```
  administrator
  ```

  Or, on Windows NT or Windows 2000, double-click the *administrator.bat* file name in File Manager or Explorer.

The Sybase Replication Agent Administrator GUI window opens.



From this window, you can access all the Administrator functions.

Creating a Replication Agent instance with Administrator

❖ **To create a new Replication Agent instance with Administrator**

1 Start Administrator.

2 Select an instance type from the list.

3 Enter an instance name that is unique on the host machine.

4 Enter a client socket port number, unique on the host machine, for the Replication Agent administration port of the new instance.

5 Click Create.

The instance appears in the List of instances. The status under Running? is no.

6 Click Done to exit the Administrator utility.

Copying a Replication Agent instance with Administrator

❖ **To copy an existing Replication Agent configuration to a new Replication Agent instance with Administrator**

1 Start Administrator.

2 Select an existing instance from the List of instances from which you want to copy the configuration file to create the new instance.

3 Enter an instance name that is unique on the host machine.

4 Enter a client socket port number, unique on the host machine, for the Replication Agent administration port of the new instance.

5 Click Copy.

Administrator lists the new instance in the List of instances.

6 Click Done to exit the Administrator utility.

Deleting a Replication Agent instance with Administrator

Before you delete a Replication Agent instance, you should:

- Remove the transaction log associated with that instance (if any). See “Removing the Replication Agent transaction log” on page 52 for more information.
- Shut down the Replication Agent application. See “Shutting down a Replication Agent instance” on page 45 for more information.
- If the Replication Agent software is installed on a Windows NT or Windows 2000 host machine, make sure that none of the files in the instance subdirectories are open and that no application or window is in any of the instance subdirectories.

Note On Windows NT and Windows 2000, if any application or window is in a file or subdirectory associated with a Replication Agent instance at the time you delete the instance, the "open" file or subdirectory will not be deleted and the deletion will be incomplete. An error message informs you of the file or subdirectory not deleted. In this event, you must verify that the file or subdirectory is not open in any application or window, and then manually delete it.

❖ **To delete a Replication Agent instance with Administrator**

1 Start Administrator.

2 Select the Replication Agent instance from the List of instances that you want to delete.

3 Click Delete.

Administrator deletes the instance from the list, unless it is running. If the instance is running, you receive an error message:

```
Cannot delete Replication Agent instance 'inst_name'
because it is currently running.
```

Note To delete an instance that is running, shut down the instance first by logging into its administrative port and using the shutdown command, then delete it. See "Shutting down a Replication Agent instance" on page 45 for more information.

If the Replication Agent instance can be deleted, a Confirm Delete dialog box appears.

4 Click OK in the Confirm Delete dialog box.

5 Click Done to exit the Administrator utility.

Starting a Replication Agent instance with Administrator

❖ **To start a Replication Agent instance with Administrator**

1 Start Administrator.

2 Select the Replication Agent instance(s) you want to start from the List of instances.

3 Click Start.

The Running column changes to `yes`.

On Windows NT or Windows 2000, a console window opens for each instance and the Replication Agent instance or instances you selected is started.

On UNIX, the Replication Agent instance or instances are started in the background. Standard error (*stderr*) output is redirected to the Replication Agent instance error log.

4 Click Done to exit the Administrator utility.

Start-up errors

If errors occur while the Replication Agent instance is starting up, they are recorded in the Replication Agent error log file (*error.log*) on UNIX, or on the command line on Windows NT or Windows 2000.

Replication Agent administration port

To gain access to the Replication Agent instance for any administrative tasks, you must log in to the Replication Agent administration port.

The Replication Agent administration port enables Sybase Replication Agent to act like an Open Server application whenever an Open Client application attempts to connect to it.

To connect to the Replication Agent administration port, you must use an Open Client (or compatible) application that sends and receives information using the Sybase Tabular Data Stream (TDS) protocol. You can use any Sybase Open Client interface utility (such as isql or SQLAdvantage) to connect to the Replication Agent administration port.

Client socket port

When you create a Replication Agent instance, you must specify a client socket port number for the administration port. Client applications use this port number to connect to the Replication Agent administration port.

Note You must specify a client socket port number that does not conflict with any existing port numbers on the Replication Agent host machine.

Creating an entry in the interfaces file

In general, Open Client applications (such as isql) require an interfaces file to identify available servers, their host machines, and their client ports. On Windows NT and Windows 2000 operating systems, the interfaces file is named *sql.ini*. On UNIX operating systems, the interfaces file is named *interfaces*.

If you want Open Client applications to be able to connect to the Replication Agent administration port, as they would to any Open Server application, you must create an entry for the Replication Agent administration port in the interfaces file.

An entry for a Replication Agent administration port in an interfaces file appears as follows:

```
[inst_name]
 query=Protocol,Host_Name,Port_Num
```

where *inst_name* is the name of the Replication Agent instance, *Protocol* is the network socket protocol used to connect with the Replication Agent administration port, *Host_Name* is the name of the Replication Agent host machine, and *Port_Num* is the client socket port number specified for the administration port when the Replication Agent instance was created.

For example, to specify an interfaces file entry for a Replication Agent instance named “NY2_RA” using the Microsoft Windows NT sockets protocol on a host named “NYHost3” with the client socket port number 10002, you would add the following lines to the interfaces file:

```
[NY2_RA]
 query=NLWNSCK,NYHost3,10002
```

Some systems require the interfaces file to be in the TLI form. If your system requires that, you must use a utility that edits the interfaces file and saves the result in a form compatible with TLI, such as sybtli or dsEdit.

After you create an entry in the interfaces file for the Replication Agent instance, you can connect to the administration port using any Open Client application that uses the interfaces file.

Logging in to the Replication Agent

To connect to the administration port of the Replication Agent instance shown in the previous example using isql, you would type the following command at the operating system prompt:

```
isql -Uusername -Ppassword -SNY2rao
```

where *username* is the administrative user ID and *password* is the corresponding password for the Replication Agent instance.

The first time you log in to a newly created Replication Agent instance, the administrative user ID is sa with no password.

Once you have successfully logged in to the administration port, you can use Replication Agent commands to administer the Replication Agent instance. See Chapter 4, “Sybase Replication Agent Command Reference” for more information.

Replication Agent administrative tasks

There are two types of administrative tasks you can perform with Sybase Replication Agent:

- Setup tasks – These are tasks that are typically performed only once when the Replication Agent software is installed. Setup tasks are described in Chapter 2, “Setting Up Sybase Replication Agent” and in the *Sybase Replication Agent Installation Guide*.
- Routine administrative tasks – These are tasks that may need to be performed from time to time during normal Replication Agent operation.

The following topics are described in this section:

- Starting and stopping Replication Agent
- Determining current Replication Agent status
- Testing network connectivity
- Maintaining the Replication Agent transaction log
- Marking and unmarking primary tables
- Enabling and disabling replication for marked tables
- Enabling and disabling replication for LOB columns
- Marking and unmarking stored procedures

- Enabling and disabling replication for stored procedures
- Configuring and tuning Replication Agent

Starting and stopping Replication Agent

Each Replication Agent instance can be started and shut down independently of other Replication Agent instances and other components in a replication system.

Starting a Replication Agent instance

There are two ways to start a Replication Agent instance:

- Using the ra command line utility
- Using the Administrator GUI

Using the ra command line utility

The Replication Agent ra command line utility allows you to start or launch a Replication Agent instance. When you start a Replication Agent instance with the ra utility, you can specify the start-up state.

Note If you do not specify a start-up state on the command line when you invoke the ra command, the Replication Agent instance goes to the *Admin* state by default.

❖ **To start a Replication Agent instance with the ra utility**

- Type the following command at the operating system prompt:

  ```
  ra -iinst_name -state
  ```

  where *inst_name* is the name of the Replication Agent instance you want to launch and *state* is the keyword for the start-up state.

See “Using the ra utility” on page 29 for more information about using the ra utility.

Using the Administrator GUI

The Administrator GUI utility allows you to start or launch a Replication Agent instance. When you start a Replication Agent instance with the Administrator GUI, the Replication Agent instance is always started in the default *Admin* start-up state.

❖ **To start a Replication Agent instance with Administrator**

1 Start Administrator.

2 Select the Replication Agent instance(s) you want to start from the List of instances.

3 Click Start.

The Running? column changes to `yes`.

4 Click Done to exit the Administrator utility.

See "Using the Administrator GUI" on page 38 for more information.

Shutting down a Replication Agent instance

To shut down a Replication Agent instance, you must log in to the instance's administration port and invoke the shutdown command.

The Replication Agent shutdown command allows you to shut down a Replication Agent instance in one of two ways:

- normal shutdown – this is the default shutdown method. A normal shutdown first quiesces the Replication Agent instance before shutting down the instance.
- immediate – this shuts down and terminates the Replication Agent instance immediately, without first quiescing. To use this option, use the immediate keyword when you invoke the shutdown command.

Note The shutdown command with no option (normal shutdown) is ignored if the Replication Agent instance is in transition from one state to another. You can invoke the shutdown command with the immediate keyword at any time, when the Replication Agent instance is in any state.

When a Replication Agent instance is quiesced, it does the following:

- Stops reading new entries in the transaction log
- Finishes processing any data already in its internal queues
- Starts its connection to the primary database if it is down and recycles its connection with the primary Replication Server
- Changes its state from *Replicating* to *Admin*

Note In the following procedure, isql is used as the Open Client application to log in to the administration port of the Replication Agent instance. You can use any other Open Client application to access the Replication Agent administration port.

❖ **To shut down a Replication Agent instance**

1 Log in to the Replication Agent administration port.

2 Once you have logged into the Replication Agent administration port, you can invoke the shutdown command:

- Use the following command to shut down the Replication Agent instance gracefully:

```
1> shutdown
2> go
```

  This command first quiesces the Replication Agent instance before shutting it down.

- Use the following command to shut down the Replication Agent instance immediately:

```
1> shutdown immediate
2> go
```

  This command shuts down and terminates the Replication Agent instance immediately, without first quiescing.

Determining current Replication Agent status

The Replication Agent status consists of the current state and activity of the Replication Agent instance.

❖ **To determine the status of a Replication Agent instance**

1 Log in to the Replication Agent administration port.

2 Once you have logged into the Replication Agent administration port, you can invoke the ra_status command:

```
1> ra_status
2> go
```

This command returns the status of the Replication Agent instance you logged into:

```
State  Action
------ --------------------------
ADMIN  Waiting for operator command
(1 row affected)
```

Replication Agent states

A Replication Agent instance can be in one of two discrete states. The Replication Agent states are:

- *Admin* – this state allows you to set or change any Replication Agent configuration parameter. In this state, the Replication Agent instance is available for any administrative operation.
- *Replicating* – this is the state the Replication Agent instance is in when it is processing data from the transaction log and sending output to the primary Replication Server (or idle while waiting for new transactions to be logged).

The default start-up state is *Admin*. The Replication Agent instance goes to the *Admin* state automatically when no start-up state is specified.

The state of a Replication Agent instance can be changed either by an external event that occurs while Replication Agent processes data or by operator intervention. An example of an external event that can cause a Replication Agent state change is a dropped connection. An example of operator intervention that can cause a Replication Agent state change is invoking a command that causes a state change.

From the time a state-changing event occurs until the moment the Replication Agent instance is actually in the new state, the Replication Agent instance is said to be in "transition." During state transition, some Replication Agent commands are ignored.

Admin state

A Replication Agent instance goes to the *Admin* state by default when it is started, when it is started with the -admin option, or when the quiesce or suspend command is invoked. In the *Admin* state, the Replication Agent instance is running, but it has no connections established to the primary database, the primary Replication Server, or the primary Replication Server RSSD.

You can perform any administrative operation while the Replication Agent instance is in the *Admin* state, including changing the value of any Replication Agent configuration parameter. Since there are no connections established when the Replication Agent instance is in the *Admin* state, you can change configuration parameters associated with communication protocols.

Note To change most of the Replication Agent configuration parameters, the Replication Agent instance must be in the *Admin* state.

Replication Agent may also return to the *Admin* state from the *Replicating* state when a network failure or other error causes a Replication Agent connection to be dropped.

To go from the *Admin* state to the *Replicating* state, you must invoke the Replication Agent resume command.

Replicating state

When a Replication Agent instance is operating normally and replicating data, it is in the *Replicating* state.

In the *Replicating* state, the Replication Agent instance is connected to the primary database and the primary Replication Server, and it is scanning its transaction log for transactions to be replicated. Actual replication (sending LTL output) does not have to be taking place for the Replication Agent instance to be in the *Replicating* state.

To go from the *Replicating* state to the *Admin* state, you can invoke either the suspend or quiesce command.

The suspend command immediately stops all Replication Agent processing, shuts down connections to the primary database and the primary Replication Server, and puts the Replication Agent instance in the *Admin* state.

The quiesce command stops all Log Reader processing of new data in the transaction log, finishes processing any change set data in the Replication Agent internal queues, shuts down connections to the primary database and the primary Replication Server, and then puts the Replication Agent instance in the *Admin* state.

Getting Replication Agent statistics

The Log Reader and Log Transfer Interface components record information about their performance whenever the Replication Agent instance is in the *Replicating* state. You can use this information to fine tune Replication Agent performance or to troubleshoot performance problems.

You can get statistical information about current Replication Agent performance by using the ra_statistics command. You can also use the ra_statistics command to reset the statistics counters.

Note Each time the Replication Agent instance enters the *Replicating* state, all the statistics counters are reset automatically.

See Chapter 4, “Sybase Replication Agent Command Reference” for more information about the Replication Agent ra_statistics command.

Testing network connectivity

Sybase Replication Agent provides a simple means of testing connections from the Replication Agent instance to the primary database and the primary Replication Server. You can invoke the Replication Agent test_connection command to send a simple connection request and confirm the connection to the primary database and the primary Replication Server.

If the connection specifications recorded in the Replication Agent configuration parameters (server name, client port address, user IDs, and passwords) are not correct, the test_connection command will return a failure message.

❖ **To verify Replication Agent connections to the primary database and primary Replication Server**

1 Log in to the Replication Agent administration port.

2 Once you have logged into the Replication Agent administration port, you can invoke the test_connection command:

```
1> test_connection
2> go
```

This command tests all the connections of the Replication Agent instance you logged into.

Note You can test a specific connection (either the primary data server or the primary Replication Server) by specifying the connection you want to test.

See Chapter 4, "Sybase Replication Agent Command Reference" for more information about the Replication Agent test_connection command.

Maintaining the Replication Agent transaction log

Log-based Replication Agents (for DB2 Universal Database) and trigger-based Replication Agents (for Informix, Microsoft SQL Server, and Oracle) have different transaction log maintenance requirements.

Log-based Replication Agents

A log-based Replication Agent uses the native transaction log maintained by the primary database to capture transactions for replication. Log-based Replication Agents also create some transaction log objects in the primary database to maintain system data for the Replication Agent, but these database objects require no routine maintenance.

Routine transaction log maintenance requirements are no different for a primary database when a log-based Replication Agent is used.

Trigger-based Replication Agents

A trigger-based Replication Agent uses its own proprietary transaction log to capture transactions in the primary database for replication. The Replication Agent transaction log consists of tables, stored procedures, and triggers that the Replication Agent creates in the primary database.

The only routine maintenance required for the trigger-based Replication Agent transaction log is truncation. If not periodically truncated, the transaction log can grow large enough to consume all the available resources allocated to the primary database.

See Chapter 4, "Sybase Replication Agent Command Reference," for more information on using the pdb_truncate_xlog command to truncate the Replication Agent transaction log.

Creating the Replication Agent transaction log

The Replication Agent instance must be running and connectivity to the primary database must be established before you can create a transaction log. See "Starting a Replication Agent instance" on page 44 for more information.

❖ **To create a Replication Agent transaction log in the primary database**

1 Log in to the Replication Agent administration port.

2 Use the pdb_xlog command to determine if a transaction log associated with this Replication Agent instance exists in the primary database:

```
pdb_xlog
```

If a transaction log does not exist for this Replication Agent instance, the pdb_xlog command returns no information about transaction log objects. Continue this procedure to create a transaction log in the primary database.

Note The pdb_xlog command looks for transaction log components based on the current value of the pdb_xlog_prefix configuration parameter. If the value of the pdb_xlog_prefix parameter changed after a transaction log was created for a Replication Agent instance, the pdb_xlog command will not find the previously created transaction log.

If a transaction log already exists for this Replication Agent instance, the pdb_xlog command returns a list of the names of the transaction log database objects. If a transaction log exists, you need not complete this procedure.

3 If you want to use a particular string for the database object name prefix of the transaction log components, use the ra_config command to set the value of the pdb_xlog_prefix parameter:

```
ra_config pdb_xlog_prefix, XXX
```

where *XXX* is a one- to three-character string that will be the new value of the pdb_xlog_prefix parameter and the prefix string that will be used in the database object names of transaction log components when the transaction log is created. The default value of the pdb_xlog_prefix parameter is ra_.

Note The value of the pdb_xlog_prefix_chars parameter specifies the non-alphabetic characters that are allowed in the prefix string (the value of the pdb_xlog_prefix parameter). The primary database server may restrict the characters that are actually allowed to be used in database object names.

You can also use the ra_config command to determine the current value of the pdb_xlog_prefix parameter:

```
ra_config pdb_xlog_prefix
```

When you invoke the ra_config command and specify a parameter with no value, it returns the current value of the specified parameter.

4 Use the pdb_xlog command to create the Replication Agent transaction log:

```
pdb_xlog create
```

Note When you invoke the pdb_xlog command with the create keyword, the command returns an error message if a transaction log already exists in the primary database using the prefix character string currently specified as the value of the pdb_xlog_prefix parameter.

After you invoke the pdb_xlog command with the create keyword, Replication Agent generates a script that, when executed, creates the transaction log base objects in the primary database.

Note You can configure Replication Agent to simply build the script, but not execute it, by setting the value of the pdb_auto_run_scripts parameter to false prior to invoking the pdb_xlog command. To complete the transaction log creation process, you have to manually execute the script.

If the log creation script executes successfully, the script is stored in a file named *create.sql* in the *rax-12_5\inst_name\scripts\xlog\installed* directory.

If the log creation script does not execute successfully, the primary database is not changed and the script is stored in a file named *create.sql* in the *rax-12_5\inst_name\scripts\xlog* directory.

Removing the Replication Agent transaction log

The Replication Agent instance must be running to remove the transaction log. See “Starting a Replication Agent instance” on page 44 for more information.

❖ **To remove the Replication Agent transaction log from the primary database**

1 Log in to the Replication Agent administration port.

2 Use the pdb_xlog command to verify that a transaction log associated with this Replication Agent instance exists in the primary database:

```
pdb_xlog
```

If a transaction log does not exist for this Replication Agent instance, the pdb_xlog command returns no information about transaction log objects. If no transaction log exists, you need not complete this procedure.

Note The pdb_xlog command looks for transaction log components based on the current value of the pdb_xlog_prefix configuration parameter. If the value of the pdb_xlog_prefix parameter changed after a transaction log was created for a Replication Agent instance, the pdb_xlog command will not find the previously created transaction log.

If a transaction log exists for this Replication Agent instance, the pdb_xlog command returns a list of the names of the transaction log database objects. Continue this procedure to remove the transaction log from the primary database.

3 Use the pdb_setreptable command to disable replication for all marked tables in the primary database:

```
pdb_setreptable all, disable
```

When you invoke the pdb_setreptable command with the all and disable keywords, Replication Agent disables replication for all marked tables in the primary database.

4 Use the pdb_setrepproc command to disable replication for all marked procedures in the primary database:

```
pdb_setrepproc all, disable
```

5 Use the pdb_setreptable command to unmark all marked tables in the primary database:

```
pdb_setreptable all, unmark
```

When you invoke the pdb_setreptable command with the all and unmark keywords, Replication Agent removes replication marking from all marked tables in the primary database.

If there are pending transactions for any table that you attempt to unmark, pdb_setreptable returns an error message. You can force an unmark, regardless of pending transactions or the table's enable status, by following the unmark keyword with the force keyword:

```
pdb_setreptable all, unmark, force
```

6 Use the pdb_setrepproc command to unmark all marked procedures in the primary database:

```
pdb_setrepproc all, unmark
```

When you invoke the pdb_setrepproc command with the all and unmark keywords, Replication Agent removes replication marking from all marked procedures in the primary database.

If there are pending transactions for any procedure that you attempt to unmark, pdb_setrepproc returns an error message. You can force an unmark, regardless of pending transactions or the procedure's enable status, by following the unmark keyword with the force keyword:

```
pdb_setrepproc all, unmark, force
```

Note Normally, if any objects in the primary database are marked for replication, you cannot remove the Replication Agent transaction log.

7 Use the pdb_xlog command to remove the Replication Agent transaction log:

```
pdb_xlog remove
```

Note When you invoke the pdb_xlog command with the remove keyword, the command returns an error message if no transaction log exists in the primary database.

After you invoke the pdb_xlog command with the remove keyword, Replication Agent generates a script that, when executed, removes the transaction log base objects from the primary database.

Note You can configure Replication Agent to simply build the script, but not execute it, by setting the value of the pdb_auto_run_scripts parameter to false prior to invoking the pdb_xlog command. To complete the transaction log removal process, you have to manually execute the script.

If the log removal script executes successfully, the script is stored in a file named *remove.sql* file in the *rax-12_5\inst_name\scripts\xlog\installed* directory.

If the log removal script does not execute successfully, the script is stored in a file named *remove.sql* in the *rax-12_5\inst_name\scripts\xlog* directory.

Forcing transaction log removal

When you invoke the pdb_xlog command with the remove keyword, Replication Agent creates a script (remove.sql) that, when executed successfully, removes all transaction log objects from the primary database.

In the event that the remove.sql script fails for some reason, some transaction log components may be removed from the primary database and some components may remain.

Note If errors cause a script execution failure, you should refer to your primary database error log(s) and the Replication Agent system log to evaluate the any errors and determine if any corrective action is necessary.

Even after you correct the cause of the failure, if you simply re-execute the remove.sql script after a failure in which some (but not all) transaction log components were removed, the script will immediately fail again because of the missing (successfully removed) transaction log components.

To avoid this problem and finish removing transaction log components after a script execution failure, you can invoke the pdb_xlog command with the remove keyword followed by the force keyword:

```
pdb_xlog remove, force
```

When you use the force keyword, the execution of the remove.sql script continues, even when errors are encountered, until the script is finished.

Marking and unmarking primary tables

To replicate transactions against tables in a primary database, the primary table (the table that contains the data affected by the transactions in the primary database) must be marked for replication and replication must be enabled for that table.

Note Marking a table for replication is separate from enabling replication for the table. If the value of the pdb_dflt_object_repl parameter is true, replication is enabled automatically at the time a table is marked. See “Enabling and disabling replication for marked tables” on page 61 for more information on enabling and disabling replication for a marked table.

If you need to temporarily suspend replication from a marked object, for example, when database maintenance operations are performed in the primary database, you can disable replication for the marked object. See “Enabling and disabling replication for marked tables” on page 61 for more information.

Log-based Replication Agents (for DB2 Universal Database) and trigger-based Replication Agents (for Informix, Microsoft SQL Server, and Oracle) have different table marking and unmarking behavior in the primary database.

Marking behavior in log-based Replication Agents

When a primary table is marked for replication with a log-based Replication Agent, such as Replication Agent for UDB, the following events occur:

- The value of the table's DATA CAPTURE option is set to DATA CAPTURE CHANGES (DB2 Universal Database).
- A row is added to the Replication Agent marked objects table in the primary database. Each row in the marked objects table lists attributes of a table marked for replication in the primary database.

If you need to change the schema of a marked table in the primary database, you must:

- Lock the table so that no new transactions occur.
- Wait for the Replication Agent to complete its processing of any transactions on the table.
- Quiesce the Replication Agent instance.
- Change the table's schema, without changing the DATA CAPTURE option.
- Unlock the table to allow user transactions to continue.
- Use the Replication Agent resume command to restart replication.

Note If you change the schema of a primary table, you may need to rematerialize the replicate table.

Unmarking behavior in log-based Replication Agents

When you unmark a table that is marked for replication with a log-based Replication Agent, such as Replication Agent for UDB, the following events occur:

- The value of the table's DATA CAPTURE option is restored to the value it had before the table was marked (DB2 Universal Database).
- The table's row in the marked objects table is deleted.

When a table is unmarked with a log-based Replication Agent, any subsequent operations that affect the data in that table are ignored.

Note In the event that a log-based Replication Agent must re-read any part of the transaction log, such as when recovering from a replication system failure, transactions that were recorded prior to a table being unmarked will not be replicated.

Marking behavior in trigger-based Replication Agents

When a primary table is marked for replication, a trigger-based Replication Agent creates several transaction log objects to support replication for that table:

- shadow table – prefixSH_xxx
- blob shadow table – prefixBSH_xxx
- shadow-row procedure – prefixSRP_xxx
- blob shadow-row procedure – prefixBSRP_xxx
- insert trigger – prefixINSTRG_xxx
- update trigger – prefixUPDTRG_xxx
- delete trigger – prefixDELTRG_xxx

where *prefix* is the one- to three-character transaction log prefix string specified by the pdb_xlog_prefix parameter and *xxx* is an alphanumeric counter string that is incremented each time an object is marked in the primary database.

Note The prefix string defined by the pdb_xlog_prefix parameter is used for all Replication Agent transaction log database objects.

For example, if the prefix string defined by the pdb_xlog_prefix parameter is ra_ (the default value) and three tables have been marked, the following names are used for the transaction log objects for the third marked table:

- shadow table – ra_SH_c
- blob shadow table – ra_BSH_c
- shadow-row procedure – ra_SRP_c
- blob shadow-row procedure – ra_BSRP_c
- insert trigger – ra_INSTRG_c
- update trigger – ra_UPDTRG_c
- delete trigger – ra_DELTRG_c

Note If triggers already exist for the primary table, the Replication Agent attempts to insert its trigger code, leaving the existing trigger code intact. You can configure the Replication Agent instance to simply build the script to mark the table, but not execute it, by setting the value of the pdb_auto_run_scripts parameter to false. To complete the table marking, you have to manually execute the script.

If you need to change the schema of a marked table in the primary database, you must first unmark the table to remove the shadow table, stored procedure, and triggers that Replication Agent creates for the primary table, then change the primary table's schema and re-mark the table.

Note If you change the schema of a primary table, you may need to rematerialize the replicate table.

Unmarking behavior in trigger-based Replication Agents

When you unmark an object that is marked for replication with a trigger-based Replication Agent, the transaction log objects (shadow table, stored procedure, and triggers) that were created to facilitate the replication for that table are removed from the primary database.

Sybase Replication Agent provides two ways to unmark an object:

- Normal – The Replication Agent first checks to see if replication is disabled for the object, then checks for any unprocessed transactions in the transaction log. If replication is not disabled for the object, or if any unprocessed transactions remain in the transaction log, the unmarking fails.
- Force – The Replication Agent unmarks the object immediately, without checking for disabled status and unprocessed transactions.

In general, the force option should be used only when you are removing the entire transaction log and discontinuing replication from the primary database, or during troubleshooting procedures.

Marking a table for replication

Use the same procedure to mark tables for replication with either a log-based Replication Agent or a trigger-based Replication Agent.

❖ **To mark a table in the primary database for replication**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setreptable command to determine if the table is already marked in the primary database.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

If the pdb_setreptable command returns information that the specified table is marked, you need not continue this procedure.

If the pdb_setreptable command returns information that the specified table is not marked, continue this procedure to mark the table for replication.

3 Use the pdb_setreptable command to mark the table for replication.

The pdb_setreptable command allows you to mark the primary table to be replicated and specify a different table name to use in the replicate database (as specified in a replication definition).

- Use the following command to mark the table for replication using a replication definition with the same table name:

```
pdb_setreptable pdb_table, mark
```

where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

- Use the following command to mark the table for replication using a replication definition with a different table name:

```
pdb_setreptable pdb_table, rep_table, mark
```

where *pdb_table* is the name of the table in the primary database that you want to mark for replication and *rep_table* is the name of the table in the with all tables named rep_table clause in the replication definition for this table.

- When marking a table for replication, you can specify that the table owner's name is sent along with the table name in the LTL. To do this, use the owner keyword after the mark keyword, as shown in the following example:

```
pdb_setreptable pdb_table, mark, owner
```

where *pdb_table* is the name of the table in the primary database that you want to mark for replication.

Note The table owner's name returned from the primary database must be the same as the table owner's name specified in the replication definition for the table.

If the value of the pdb_dflt_object_repl parameter is true, the table marked for replication with the pdb_setreptable command is ready for replication after you invoke the pdb_setreptable command successfully.

If the value of the pdb_dflt_object_repl parameter is true, you can skip step 4 in this procedure.

Note The default value of the pdb_dflt_object_repl parameter is true.

If the value of the pdb_dflt_object_repl parameter is false, you must enable replication for the table before replication can take place.

4 Use the pdb_setreptable command to enable replication for the marked table.

```
pdb_setreptable pdb_table, enable
```

where *pdb_table* is the name of the marked table in the primary database for which you want to enable replication.

After replication is enabled for the table, you can begin replicating transactions that affect data in that table.

Unmarking a table

Use the same procedure to unmark tables for replication with either a log-based Replication Agent or a trigger-based Replication Agent.

❖ **To unmark a table in the primary database**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setreptable command to confirm that the table is marked in the primary database.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the table in the primary database that you want to unmark.

If the pdb_setreptable command returns information that the specified table is marked, continue this procedure to unmark the table.

If the pdb_setreptable command does not return information that the specified table is marked, you need not continue this procedure.

3 Use the pdb_setreptable command to disable replication from the table.

```
pdb_setreptable pdb_table, disable
```

where *pdb_table* is the name of the table in the primary database that you want to unmark.

4 Use the pdb_setreptable command to remove the replication marking from the table.

```
pdb_setreptable pdb_table, unmark
```

where *pdb_table* is the name of the table in the primary database that you want to unmark.

If you need to force the unmark, you can use the following command:

```
pdb_setreptable pdb_table, unmark, force
```

5 Use the pdb_setreptable command to confirm that the table is no longer marked for replication.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the table in the primary database that you unmarked.

Note You can unmark all marked objects in the primary database by invoking the pdb_setreptable command with the all keyword.

Enabling and disabling replication for marked tables

If you need to temporarily suspend replication from a table, you can use the pdb_setreptable command to disable replication for the marked table. When you are ready to resume replication from the marked table, you can use the pdb_setreptable command to enable replication.

To replicate transactions in the primary database, the primary table that contains the data affected by the transactions must be marked for replication and replication must be enabled for that table.

Note Marking a table for replication is separate from enabling replication for the table. See “Marking and unmarking primary tables” on page 55 for more information on marking a table for replication.

The Replication Agent marked objects table contains an entry for each marked table in the primary database. Each marked table row contains a flag indicating whether replication is enabled or disabled for the marked table.

When replication is disabled for a marked object, the marking infrastructure remains in place, but no transactions for that object are captured in the transaction log.

Note If you need to change the schema of a marked table in the primary database, you must first unmark the table to remove the transaction log objects that Replication Agent creates for the primary table. See “Marking and unmarking primary tables” on page 55 for more information.

Enabling replication for tables

Use the same procedure to enable replication for marked tables with either a log-based Replication Agent or a trigger-based Replication Agent.

❖ **To enable replication for a marked table**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setreptable command to verify that replication is disabled for the table.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the marked table you want to enable replication for.

If the pdb_setreptable command returns information that the table is marked and has replication disabled, continue this procedure to enable replication for the table.

Note A table must be marked for replication before replication can be enabled or disabled for the table.

3 Use the pdb_setreptable command to enable replication for the table.

```
pdb_setreptable pdb_table, enable
```

where *pdb_table* is the name of the marked table in the primary database for which you want to enable replication.

After replication is enabled for the table, any transaction that affects the data in that table will be captured for replication.

4 You can use the pdb_setreptable command again to verify that replication is now enabled for the table.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the marked table for which you want to verify that replication is enabled.

Disabling replication for tables

Use the same procedure to disable replication for marked tables with either a log-based Replication Agent or a trigger-based Replication Agent.

❖ **To disable replication for a marked table**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setreptable command to verify that replication is enabled for the table.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the marked table you want to disable replication for.

If the pdb_setreptable command returns information that the table is marked and has replication enabled, continue this procedure to disable replication for the table.

Note A table must be marked for replication before replication can be enabled or disabled for the table.

---

3 Use the pdb_setreptable command to disable replication for the table.

```
pdb_setreptable pdb_table, disable
```

where *pdb_table* is the name of the marked table in the primary database for which you want to disable replication.

After replication is disabled for the table, transactions that affect the data in that table will not be captured for replication until replication is enabled again.

4 You can use the pdb_setreptable command again to verify that replication is now disabled for the table.

```
pdb_setreptable pdb_table
```

where *pdb_table* is the name of the marked table for which you want to verify that replication is disabled.

Enabling and disabling replication for LOB columns

In this document, all columns that contain large object (LOB) datatypes are referred to as LOB columns, regardless of the actual datatype name used by the primary database vendor. To replicate transactions that affect a LOB column, replication must be enabled for that column.

You must enable replication for each LOB column you want to replicate, in addition to marking and enabling replication for the table that contains the LOB column.

If the value of the pdb_dflt_column_repl parameter is true, replication is enabled automatically for all LOB columns in a table at the time the table is marked. If the value of the pdb_dflt_column_repl parameter is false, replication is not enabled automatically for any LOB columns in a table at the time the table is marked. See “Marking and unmarking primary tables” on page 55 for more information on marking a table for replication.

When a table is marked for replication and replication is enabled for that table but not for a LOB column in that table, any part of a transaction that affects the LOB column is not replicated. The portion of a transaction that affects all other non-LOB columns is replicated if the table is marked for replication and replication is enabled for the table.

When replication is enabled for a LOB column, Replication Agent makes an entry in the prefixBLOB_COLUMNS_ table to support replication for that column.

When Replication Agent triggers processes a transaction that affects a LOB column, the LOB data is not stored in the transaction log because of its possible size. Instead, the Replication Agent Log Reader component reads the LOB data directly from the primary database at the time it processes the transaction.

Because of the way Replication Agent processes the LOB column data when replicating transactions, it is possible to compromise transaction integrity. For example, if two transactions change the data in a LOB column and the Log Reader doesn't process the first transaction until after the second transaction has been committed, when the LOB data is read from the primary database, the value of that data is the result of the second transaction. In this event, the value of the LOB data in the first transaction is never sent to the replicate database. After the second transaction is processed by the Log Reader, the primary and replicate databases will be synchronized again, but for a period of time between processing the first and second transactions, the replicate database contains data that does not match the originating transaction.

This problem occurs only when a LOB column is changed more than once by a sequence of transactions. The period of time over which the problem exists could be significant if the replication system throughput is slow or if a replication system component fails. As soon as the last transaction that changes the LOB column is processed at the replicate site, the problem will be corrected.

Enabling replication for LOB columns

❖ **To enable replication for a LOB column in a marked table**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepcol command to verify that replication is disabled for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column.

If the pdb_setrepcol command returns information that the LOB column has replication disabled, continue this procedure to enable replication for the column.

Note The table that contains the LOB column must be marked for replication before replication can be enabled or disabled for a LOB column.

---

3 Use the pdb_setrepcol command to enable replication for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col, enable
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column for which you want to enable replication.

After replication is enabled for the LOB column (and if replication is enabled for the table that contains the column), any transaction that affects the data in that column will be replicated.

4 You can use the pdb_setrepcol command again to verify that replication is now enabled for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column for which you want to verify that replication is enabled.

Disabling replication for LOB columns

❖ **To disable replication for a LOB column in a marked table**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepcol command to verify that replication is enabled for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column you want to disable replication for.

If the pdb_setrepcol command returns information that the LOB column has replication enabled, continue this procedure to disable replication for the column.

Note The table that contains the LOB column must be marked for replication before replication can be enabled or disabled for a LOB column.

---

3 Use the pdb_setrepcol command to disable replication for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col, disable
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column for which you want to disable replication.

After replication is disabled for the LOB column, transactions that affect the data in that column will not be replicated unless replication is enabled for that column again.

4 You can use the pdb_setrepcol command again to verify that replication is now disabled for the LOB column.

```
pdb_setrepcol pdb_table, pdb_col
```

where *pdb_table* is the name of the marked table that contains the LOB column and *pdb_col* is the name of the LOB column for which you want to verify that replication is disabled.

Marking and unmarking stored procedures

Sybase Replication Agent supports Replication Server function replication by replicating the invocation of stored procedures in the primary database.

Note In this document, the terms *function* and *stored procedure* are synonyms.

Sybase Replication Agent can replicate both *applied functions* and *request functions*. Applied functions are stored procedures that are executed in the primary database and generate transactions that affect data in the primary database. Request functions are stored procedures that are invoked in one database (for example, a replicate database), then executed in another database (for example, a primary database). Replication Agent does not distinguish between these two function types.

In order to replicate a stored procedure invoked in a primary database, the stored procedure must be marked for replication and replication must be enabled for that stored procedure. (This is analogous to marking and enabling replication for tables.)

Note Marking a stored procedure for replication is separate from enabling replication for the stored procedure. If the value of the pdb_dflt_object_repl parameter is true, replication is enabled automatically at the time a stored procedure is marked. See “Enabling and disabling replication for stored procedures” on page 71 for more information on enabling and disabling replication for a marked stored procedure.

If a marked stored procedure performs operations that affect a marked table, the operations that affect the marked table are not captured for replication, only the invocation of the marked stored procedure is replicated.

When you mark a stored procedure for replication, Replication Agent creates a shadow-row procedure for that stored procedure.

Replication Agent also modifies the marked stored procedure as follows:

- Inserts a new first step to execute the associated shadow- row procedure
- Inserts a new last step to again execute the shadow-row procedure with different parameters.

If you need to temporarily suspend replication of a marked stored procedure, for example, when database maintenance operations are performed in the primary database, you can disable replication for the stored procedure. See "Enabling and disabling replication for stored procedures" on page 71 for more information.

When you unmark an object that has been marked for replication, the transaction log objects that were created to facilitate the replication for that object are removed from the primary database.

Note Procedure replication is not implemented for DB2 Universal Database.

For more information on the Replication Server function replication feature, see the Replication Server *Administration Guide*.

Marking a stored procedure for replication

When you mark a stored procedure for replication, Replication Agent creates the transaction log objects that capture the stored procedure invocation in the transaction log.

❖ **To mark a stored procedure for replication**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepproc command to determine if the stored procedure is already marked in the primary database.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication.

If the pdb_setrepproc command returns information that the specified stored procedure is marked, you need not continue this procedure.

If the pdb_setrepproc command returns information that the specified stored procedure is not marked, continue this procedure to mark the stored procedure for replication.

3 Use the pdb_setrepproc command to mark the stored procedure for replication.

The pdb_setrepproc command allows you to mark the primary stored procedure to be replicated and specify a different stored procedure name to use in the replicate database (as specified in a function replication definition).

- Use the following command to mark the stored procedure for replication using a function replication definition with the same stored procedure name:

  ```
  pdb_setrepproc pdb_proc, mark
  ```

  where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication.

- Use the following command to mark the stored procedure for replication using a function replication definition with a different stored procedure name:

  ```
  pdb_setrepproc pdb_proc, rep_proc, mark
  ```

  where *pdb_proc* is the name of the stored procedure in the primary database that you want to mark for replication and *rep_proc* is the name of the stored procedure in the function replication definition for this stored procedure.

  If the value of the pdb_dflt_object_repl parameter is true, the stored procedure marked for replication with the pdb_setrepproc command is ready for replication after you invoke the pdb_setrepproc command successfully.

If the value of the pdb_dflt_object_repl parameter is true, you can skip step 4 in this procedure.

Note The default value of the pdb_dflt_object_repl parameter is true.

If the value of the pdb_dflt_object_repl parameter is false, you must enable replication for the stored procedure before replication can take place.

4 Use the pdb_setrepproc command to enable replication for the marked stored procedure.

```
pdb_setrepproc pdb_proc, enable
```

where *pdb_proc* is the name of the marked stored procedure for which you want to enable replication.

After replication is enabled for the stored procedure, you can begin replicating invocations of that stored procedure in the primary database.

Unmarking a stored procedure

When you unmark a stored procedure, Replication Agent removes the transaction log objects that were created when the stored procedure was marked.

❖ **To unmark a stored procedure**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepproc command to confirm that the stored procedure is marked in the primary database.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the stored procedure that you want to unmark.

If the pdb_setrepproc command returns information that the specified stored procedure is marked, continue this procedure to unmark the stored procedure.

If the pdb_setrepproc command does not return information that the specified stored procedure is marked, you need not continue this procedure.

3 Use the pdb_setrepproc command to disable replication of the stored procedure.

```
pdb_setrepproc pdb_proc, disable
```

where *pdb_proc* is the name of the stored procedure that you want to unmark.

4 Use the pdb_setrepproc command to remove the replication marking from the stored procedure.

```
pdb_setrepproc pdb_proc, unmark
```

where *pdb_proc* is the name of the stored procedure that you want to unmark.

If you need to force the unmark, you can use the following command:

```
pdb_setrepproc pdb_proc, unmark, force
```

5 Use the pdb_setrepproc command to confirm that the stored procedure is no longer marked for replication.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the stored procedure in the primary database that you unmarked.

Note You can unmark all marked stored procedures in the primary database by invoking the pdb_setrepproc command with the all keyword.

Enabling and disabling replication for stored procedures

If you need to temporarily suspend replication of a stored procedure, you can use the pdb_setrepproc command to disable replication for the marked stored procedure. When you are ready to resume replication of the marked stored procedure, you can use the pdb_setrepproc command to enable replication.

Note Procedure replication is not implemented for DB2 Universal Database.

To replicate invocations of a stored procedure in the primary database, the stored procedure must be marked for replication and replication must be enabled for that stored procedure.

Marking a stored procedure for replication is separate from enabling replication for the stored procedure. See “Marking and unmarking stored procedures” on page 67 for more information on marking a stored procedure for replication.

Enabling replication for stored procedures

❖ **To enable replication for a marked stored procedure**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepproc command to verify that replication is disabled for the stored procedure.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the marked stored procedure you want to enable replication for.

If the pdb_setrepproc command returns information that the stored procedure is marked and has replication disabled, continue this procedure to enable replication for the stored procedure.

Note A stored procedure must be marked for replication before replication can be enabled or disabled for the stored procedure.

3 Use the pdb_setrepproc command to enable replication for the stored procedure.

```
pdb_setrepproc pdb_proc, enable
```

where *pdb_proc* is the name of the marked stored procedure for which you want to enable replication.

After replication is enabled for the stored procedure, any invocation of that stored procedure will be replicated.

4 You can use the pdb_setrepproc command again to verify that replication is now enabled for the stored procedure.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the marked stored procedure for which you want to verify that replication is enabled.

Disabling replication for stored procedures

❖ **To disable replication for a marked stored procedure**

1 Log in to the Replication Agent administration port.

2 Use the pdb_setrepproc command to verify that replication is enabled for the stored procedure.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the marked stored procedure you want to disable replication for.

If the pdb_setrepproc command returns information that the stored procedure is marked and has replication enabled, continue this procedure to disable replication for the stored procedure.

Note A stored procedure must be marked for replication before replication can be enabled or disabled for that stored procedure.

3 Use the pdb_setrepproc command to disable replication for the stored procedure.

```
pdb_setrepproc pdb_proc, disable
```

where *pdb_proc* is the name of the marked stored procedure for which you want to disable replication.

After replication is disabled for the stored procedure, any invocation of that stored procedure will not be captured for replication until replication is enabled again.

4 You can use the pdb_setrepproc command again to verify that replication is now disabled for the stored procedure.

```
pdb_setrepproc pdb_proc
```

where *pdb_proc* is the name of the marked stored procedure for which you want to verify that replication is disabled.

Configuring and tuning Replication Agent

The performance of Sybase Replication Agent can be tuned or optimized by adjusting some of the Replication Agent configuration parameters. The following section describes recommended values for various Replication Agent configuration parameters.

To set or change a Replication Agent configuration parameter, use the ra_config command.

Because Replication Agent overwrites its entire configuration file whenever ra_config or ra_set_login is invoked, Sybase recommends that you do not hand edit the configuration file. Furthermore, Replication Agent reads the configuration file only once at start up. You must use the ra_config command if you want the new configuration parameter to take effect immediately.

See Chapter 4, “Sybase Replication Agent Command Reference” for more information on using the ra_config command.

See Chapter 5, “Sybase Replication Agent Configuration Parameters” for more information about configuration parameters.

Replication Agent recommended configuration

Sybase recommends the values for Replication Agent configuration parameters listed in Table 3-1.

There are no recommended values for the configuration parameters not listed in the following table.

Table 3-1: Recommended configuration parameters

Configuration Parameter	Recommended Value
column_compression	true
compress_ltl_syntax	true
dump_batch_timeout	5
log_wrap	500
lr_update_trunc_point	1000
lti_batch_mode	true
lti_max_buffer_size	5000
lti_update_trunc_point	1000
ltl_origin_time_required	false
maint_cmds_to_skip_before_update	1000
max_ops_per_scan	1000
pdb_convert_datetime	false
pds_retry_count	5
pds_retry_timeout	10
ra_retry_count	2
ra_retry_timeout	10
rs_retry_count	5
rs_retry_timeout	10
scan_sleep_increment	5
scan_sleep_max	60
skip_ltl_errors	false
structured_tokens	false
truncation_type	locator_update
use_rssd	true

Note Log truncation is not available with log-based Replication Agents (for DB2 Universal Database). Therefore, the truncation_type parameter recommendation in Table 3-1 does not apply to log-based Replication Agents.

CHAPTER 4

Sybase Replication Agent Command Reference

This chapter describes all the Replication Agent commands.

Table 4-1: Replication Agent commands

Command Name	Description
log_system_name	Returns the full path to the Replication Agent system log file.
log_trace_name	Returns the full path to the Replication Agent trace log file.
pdb_capabilities	Returns a list of the Replication Agent capabilities.
pdb_date	Returns the current date and time from the primary data server.
pdb_execute_sql	Executes the specified SQL statement in the current database.
pdb_gen_id	Returns the current value of the database generation ID or updates the value of the database generation ID.
pdb_get_columns	Returns a list of all the columns in the specified table.
pdb_get_databases	Returns a list of all the databases in the primary data server.
pdb_get_primary_keys	Returns a list of all the columns that make up the primary keys in the specified table.
pdb_get_procedure_parms	Returns a list of the parameters for the specified procedure.
pdb_get_procedures	Returns a list of all the procedures in the specified database.
pdb_get_sql_database	Returns the name of the database specified for SQL statement execution.
pdb_get_tables	Returns a list of all the tables in the specified database.
pdb_set_sql_database	Specifies the database to be used for SQL statement execution.

Command Name	Description
pdb_setrepcol	Returns replication marking status; enables or disables replication for all marked columns or a specified column.
pdb_setrepproc	Returns replication marking status; unmarks all marked procedures or a specified procedure; enables or disables replication for all marked procedures or a specified procedure; marks a specified procedure for replication.
pdb_setreptable	Returns replication marking status; unmarks all marked tables or a specified table; enables or disables replication for all marked tables or a specified table; marks a specified table for replication.
pdb_truncate_xlog	Truncates the Replication Agent transaction log.
pdb_version	Returns the type and version of the primary database server.
pdb_xlog	Returns names of transaction log objects, creates transaction log base objects in the primary database, or removes transaction log base objects from the primary database.
quiesce	Stops current Log Reader activity, processes data in internal queues, drops connections, and puts Replication Agent in the *Admin* state.
ra_config	Returns help information for configuration parameter(s) or sets the value of a configuration parameter.
ra_date	Returns the current date and time from the Replication Agent server.
ra_dump	Places a dump marker in the Replication Agent transaction log.
ra_help	Returns help information for Replication Agent command(s).
ra_locator	Returns the current value of the LTM Locator stored by Replication Agent, zeroes the current LTM Locator, or retrieves a new LTM Locator from Replication Server.
ra_maintid	Returns the Replication Server maintenance ID for the Replication Agent connection.

Command Name	Description
ra_marker	Places a marker in the Replication Agent transaction log.
ra_set_login	Sets the Replication Agent admin user login and password.
ra_statistics	Returns statistics for either a specified Replication Agent component or all components, or resets statistics for all components.
ra_status	Returns the current Replication Agent state.
ra_version	Returns the Replication Agent version.
ra_version_all	Returns Replication Agent, primary data server, and Replication Server versions.
resume	Starts replication for the currently active log, puts Replication Agent in the *Replicating* state.
shutdown	Shuts down Replication Agent.
suspend	Immediately stops all Log Reader activity, drops connections, and puts Replication Agent in the *Admin* state.
test_connection	Tests Replication Agent connections.
trace	Returns current trace flag settings or changes a specified trace flag.

The remaining sections in this chapter describe each Replication Agent command in detail.

log_system_name

Description

Returns the full path to the Replication Agent system log file for the current Replication Agent instance.

Syntax

log_system_name

Usage

- When a Replication Agent instance is created, a log directory is created as part of the instance directory structure and the value of the log_directory parameter points to that directory, for example:

  ```
  C:\%SYBASE%\rax-12_5\inst_name\log\
  ```

  where *%SYBASE%* is the Replication Agent installation directory and *inst_name* is the name of the Replication Agent instance.

If you specify a valid directory path as the value of the log_directory parameter, Replication Agent places its system and error log files in the directory you specify.

- The Replication Agent system log file contains output from the trace points identified by the SYSTEM trace flags. A list of the SYSTEM trace flags is provided in the description of the trace command beginning on page 138.
- The log_system_name command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also: log_trace_name, ra_config, trace

log_trace_name

Description: Returns the full path to the Replication Agent trace log file for the current Replication Agent instance.

Syntax: log_trace_name

Usage

- The default path of the Replication Agent trace log file is:

  ```
  C:\%SYBASE%\rax-12_5\inst_name\log\trace.log
  ```

  where *%SYBASE%* is the Replication Agent installation directory and *inst_name* is the name of the Replication Agent instance.
- The Replication Agent trace log file contains output from the trace points identified by the TRACE trace flags. A list of the TRACE trace flags is provided in the description of the trace command beginning on page 138.
- The log_trace_name command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also: log_system_name, trace

pdb_capabilities

Description: Returns a list of the Replication Agent capabilities.

Syntax: pdb_capabilities

Usage

- When pdb_capabilities is invoked, it returns a list of the capabilities of the Replication Agent instance.

- The purpose of the pdb_capabilities command is to support the Replication Server Manager.
- The pdb_capabilities command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

pdb_date

Description — Returns the current date and time from the primary database server.

Syntax — pdb_date

Examples

```
pdb_date
```

This command returns the current date and time from the primary database server, as shown below:

```
Current PDB Date
 ----------------------------
     Nov 30 2000 12:09:47.310
(1 row affected)
```

Usage

- When pdb_date is invoked, it returns the current date and time from the primary database server in the form of a Sybase datetime datatype.
- The pdb_date command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also — ra_date

pdb_execute_sql

Description — Executes a SQL statement in the current database at the primary database server.

Syntax — pdb_execute_sql *statement*

Parameters

statement
A SQL statement to be executed in the current database at the primary database server.

Usage

- The pdb_execute_sql command allows simple SQL queries to be executed in a database.

- The SQL statement specified as a parameter of the pdb_execute_sql command must be a single SQL command enclosed in double quotes. For example:

  ```
  pdb_execute_sql "select * from Authors"
  ```

  No command terminator is required. No syntax or other validation is performed; the string is passed directly to the current database.
- When pdb_execute_sql is invoked, it executes the specified SQL statement against the current database.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).
- To set or change the current database, use the pdb_set_sql_database command.

Note If the pdb_set_sql_database command has not been invoked to set the current database, the pdb_execute_sql command executes the SQL query against the primary database to which the Replication Agent instance is connected.

- Any results returned from execution of the SQL statement are passed to the client via the Replication Agent administration port.
- To find out the name of the current database, use the pdb_get_sql_database command. If the pdb_set_sql_database command has not been invoked to set the current database, the pdb_get_sql_database command returns the default current database.
- The pdb_execute_sql command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_get_sql_database, pdb_set_sql_database

pdb_gen_id

Description Returns the current value of the database generation ID or updates the value of the database generation ID.

Syntax pdb_gen_id [*number*]

Parameters

number

This is the value of the new database generation ID to be used when the database generation ID is updated.

Examples

Example 1

```
pdb_gen_id
```

This command returns the current value of the database generation ID, as shown below:

```
Generation ID
 -------------
 0
 (1 row affected)
```

Example 2

```
pdb_gen_id 10
```

This command updates the value of the database generation ID.

Usage

- When pdb_gen_id is invoked with no option, it returns the current value of the database generation ID stored in the Replication Agent transaction log system table.
- When pdb_gen_id is invoked with the *number* option, it updates the value of the database generation ID in the Replication Agent transaction log system table. Changing the database generation ID takes effect immediately.
- The database generation ID is the first two bytes of the origin queue ID. The database generation ID is used by Replication Server to support recovery operations, which may require Replication Agent to re-send transactions.

  During recovery, if Replication Agent must re-send operations that the primary Replication Server has already processed, the database generation ID can be changed to prevent the primary Replication Server from recognizing the information as already processed.

  For more information about the origin queue ID, see the appendix for your specific primary database server at the end of this book.
- If the Replication Agent transaction log does not exist, the pdb_gen_id command returns an error message.
- The pdb_gen_id command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also

ra_locator, pdb_truncate_xlog

pdb_get_columns

Description

Returns a list of columns in the specified table in the current database at the primary database server.

Syntax

pdb_get_columns [*ownername*, *tablename*[, *colname*]]

Parameters

ownername

The name of the owner of the table specified in the *tablename* parameter. This parameter can be delimited with quote characters to specify character case.

tablename

The name of the table in the current database for which information is returned. This parameter can be delimited with quote characters to specify character case.

colname

The name of the column for which information is returned. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_columns is invoked, it returns a result set that lists the column (or columns) in the specified table owned by the specified owner in the current database.
- When pdb_get_columns is invoked with no options, it returns a result set that lists all the columns in all the tables owned by all the owners in the current database.
- The pdb_get_columns command accepts the % character as a wild card in the *ownername*, *tablename*, and *colname* options.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

  Note If a current database is not set with the pdb_set_sql_database command, the pdb_get_columns command returns results from the primary database to which the Replication Agent instance is connected.

- To set or change the current database, use the pdb_set_sql_database command.
- To find out the name of the current database, use the pdb_get_sql_database command.

- The pdb_get_columns command returns 0 rows if the specified table (with the specified owner) does not exist in the current database or if the specified column does not exist in the specified table.
- The pdb_get_columns command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also: pdb_get_databases, pdb_get_primary_keys, pdb_get_procedure_parms, pdb_get_procedures, pdb_get_tables

pdb_get_databases

Description: Returns a list of all databases in the primary data server.

Syntax: pdb_get_databases

Usage:

- When pdb_get_databases is invoked, it returns a result set that lists all the databases in the primary data server.

  Note System databases may or may not be returned by some primary database servers when the pdb_get_databases command is invoked. See the appendix for your specific primary database server at the end of this book for more information.

- If the primary database server does not support multiple databases, no results are returned by the pdb_get_databases command.
- The pdb_get_databases command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also: pdb_get_columns, pdb_get_primary_keys, pdb_get_procedure_parms, pdb_get_procedures, pdb_get_tables

pdb_get_primary_keys

Description: Returns a list of all columns that are defined as primary keys in the specified table in the current database at the primary database server.

Syntax: pdb_get_primary_keys *ownername*, *tablename*

Parameters

ownername
The name of the owner of the table specified in the *tablename* parameter. This parameter can be delimited with quote characters to specify character case.

tablename
The name of the table in the current database for which primary key column information is returned. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_primary_keys is invoked, it returns a result set that lists all the columns that are defined as primary keys in the specified table with the specified owner in the current database.
- The pdb_get_primary_keys command accepts the % character as a wild card in the *ownername* option, but not in the *tablename* option.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

  Note If a current database is not set with the pdb_set_sql_database command, the pdb_get_primary_keys command returns results from the primary database to which the Replication Agent instance is connected.

- To set or change the current database, use the pdb_set_sql_database command.
- To find out the name of the current database, use the pdb_get_sql_database command.
- The pdb_get_primary_keys command returns 0 rows if the specified table with the specified owner does not exist in the current database.
- The pdb_get_primary_keys command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also

pdb_get_columns, pdb_get_databases, pdb_get_procedure_parms, pdb_get_procedures, pdb_get_tables

pdb_get_procedure_parms

Description

Returns a list of all parameters for the specified procedure in the current database at the primary database server.

Syntax

pdb_get_procedure_parms [*ownername*, *procname*, *paramname*]

Parameters

ownername

The name of the owner of the procedure specified in the *procname* parameter. This parameter can be delimited with quote characters to specify character case.

procname

The name of the procedure in the current database for which information is returned. This parameter can be delimited with quote characters to specify character case.

paramname

The name of the procedure parameter for which information is returned. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_procedure_parms is invoked, it returns a result set that lists all the parameters for the specified procedure(s) in the current database.
- When pdb_get_procedure_parms is invoked with no options, it returns a result set that lists all the parameters for all the procedures in the current database.
- The pdb_get_procedure_parms command accepts the % character as a wild card in both the *ownername* and *procname* options.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

  Note If a current database is not set with the pdb_set_sql_database command, the pdb_get_procedure_parms command returns results from the primary database to which the Replication Agent instance is connected.

- To set or change the current database, use the pdb_set_sql_database command.
- To find out the name of the current database, use the pdb_get_sql_database command.
- The pdb_get_procedure_parms command returns 0 rows if the specified procedure (with the specified owner) does not exist in the current database.
- The pdb_get_procedure_parms command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_get_columns, pdb_get_databases, pdb_get_primary_keys, pdb_get_procedures, pdb_get_tables

pdb_get_procedures

Description Returns a list of all procedures in the current database at the primary database server.

Syntax pdb_get_procedures [*ownername*, *procname*]

Parameters

ownername
The name of the owner of the procedure specified in the *procname* parameter. This parameter can be delimited with quote characters to specify character case.

procname
The name of the procedure in the current database for which information is returned. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_procedures is invoked, it returns a result set that lists all the procedures (with the specified owner) in the current database.
- When pdb_get_procedures is invoked with no options, it returns a result set that lists all the procedures in the current database.
- The pdb_get_procedures command accepts the % character as a wild card in both the *ownername* and *procname* options.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

  Note If a current database is not set with the pdb_set_sql_database command, the pdb_get_procedures command returns results from the primary database to which the Replication Agent instance is connected.

- To set or change the current database, use the pdb_set_sql_database command.
- To find out the name of the current database, use the pdb_get_sql_database command.
- The pdb_get_procedures command returns 0 rows if the specified procedure (with the specified owner) does not exist in the current database.

- The pdb_get_procedures command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_get_columns, pdb_get_databases, pdb_get_primary_keys, pdb_get_procedure_parms, pdb_get_tables

pdb_get_sql_database

Description Returns the name of the current database, if any.

Syntax pdb_get_sql_database

Usage

- When pdb_get_sql_database is invoked, it returns the name of the current database.
- If a database has not been specified as the current database with the pdb_set_sql_database command, the pdb_get_sql_database command returns the default current database.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

Note If a database has not been specified as the current database with the pdb_set_sql_database command, the pdb_execute_sql command executes the SQL query against the primary database (the default current database).

- To set or change the current database, use the pdb_set_sql_database command.
- The pdb_get_sql_database command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_execute_sql, pdb_set_sql_database

pdb_get_tables

Description Returns a list of all tables in the current database at the primary database server, except system tables.

Syntax pdb_get_tables [*ownername*, *tablename*]

Parameters

ownername
The name of the owner of the table specified in the *tablename* parameter. This parameter can be delimited with quote characters to specify character case.

tablename
The name of the table in the current database for which information is returned. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_tables is invoked, it returns a result set that lists the specified table with the specified owner in the current database.
- When pdb_get_tables is invoked with no options, it returns a result set that lists all the tables in the current database.

Note System tables may or may not be returned by some primary database servers when the pdb_get_tables command is invoked. See the appendix for your specific primary database server at the end of this book for more information.

- The pdb_get_tables command accepts the % character as a wild card in the both the *ownername* and *tablename* options.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter).

Note If a current database is not set with the pdb_set_sql_database command, the pdb_get_tables command returns results from the primary database to which the Replication Agent instance is connected.

- To set or change the current database, use the pdb_set_sql_database command.
- To find out the name of the current database, use the pdb_get_sql_database command.
- The pdb_get_tables command returns 0 rows if the specified table (with the specified owner) does not exist in the current database.
- The pdb_get_tables command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also

pdb_get_columns, pdb_get_databases, pdb_get_primary_keys, pdb_get_procedure_parms, pdb_get_procedures

pdb_set_sql_database

Description — Sets the current database to be used for SQL statement execution.

Syntax — pdb_set_sql_database *database*

Parameters — *database*

The name of the database in the primary data server against which SQL statements can be executed. This parameter can be delimited with quote characters to specify character case.

Usage

- When pdb_get_sql_database is invoked, it sets the database in which SQL statements can be executed.

  Note Some database servers and communication drivers do not allow you to change the current database.

- The database name you specify with the pdb_get_sql_database command is not validated. If you specify an invalid database name, no error is returned until the pdb_execute_sql command is invoked.
- To determine which database is currently specified for SQL statement execution, use the pdb_get_sql_database command. If a database has not been specified for SQL statement execution with the pdb_set_sql_database command, the pdb_get_sql_database command returns the default current database.
- The default current database is the primary database to which the Replication Agent instance is connected (as specified by the Replication Agent pds_database_name configuration parameter). If a database has not been specified for SQL statement execution, the pdb_execute_sql command executes the SQL query against the primary database.
- The pdb_set_sql_database command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also — pdb_execute_sql, pdb_get_sql_database

pdb_setrepcol

Description — Returns LOB column replication status; enables or disables replication for LOB columns within marked tables.

Syntax

To return replication status of all columns in all tables or all columns in a specific table:

pdb_setrepcol [*tablename*|enable|disable]

To return replication status of a specific column in a specific table:

pdb_setrepcol *tablename, colname*

To enable or disable all LOB columns in all marked tables:

pdb_setrepcol all, {enable|disable [, force]}

To enable, or disable replication for a specified LOB column:

pdb_setrepcol *tablename, colname*, {enable|disable [, force]}

Parameters

tablename

The name of the user table in the primary database that contains the column specified in the *colname* parameter.

The *tablename* parameter can be owner-qualified (include the owner name), with each element separated by a period. For example:

```
owner.table
```

This parameter can be delimited with quote characters to specify the character case.

If mixed case (uppercase and lowercase) is required, the name must be delimited. For example:

```
"Owner".table
"Owner"."Table"
```

Each mixed case element of the *tablename* parameter must be delimited separately, as illustrated in the previous example.

Note If you must use an object name case that does not match the value of the ltl_character_case parameter, the object name must be delimited.

If an object name contains any non-alphanumeric characters, such as spaces, periods, and so forth, it must be delimited with quote characters. For example:

```
"table name"
owner."table name"
```

If an object name contains a period, it must be *both* owner-qualified and delimited with quote characters. For example:

```
owner."table.name"
"table.owner"."table.name"
```

colname

The name of a LOB column in the user table specified in the *tablename* parameter.

This parameter can be delimited with quote characters to specify the character case.

If mixed case (uppercase and lowercase) is required, the name must be delimited. For example:

```
"Colname"
"COLname"
```

Note If you must use a column name case that does not match the value of the ltl_character_case parameter, the column name must be delimited.

---

all

A keyword that refers to all LOB columns in marked tables in the primary database. By using the all keyword, you can apply an enable, or disable operation to all LOB columns in marked tables.

enable

A keyword that refers to enabling replication for LOB columns.

disable

A keyword that refers to disabling replication for LOB columns.

force

A keyword that refers to forcing replication to be disabled for LOB columns.

When the force keyword follows the disable keyword, the pdb_setrepcol command immediately disables replication for the specified LOB column, without first checking for pending operations in the transaction log. When the force keyword follows the disable keyword and the all keyword, the pdb_setrepcol command immediately disables replication for all marked LOB columns in marked tables in the primary database, regardless of any pending operations in the transaction log.

Examples

Example 1

```
pdb_setrepcol
```

This command returns replication information for all enabled LOB columns in marked tables in the primary database.

Example 2

```
pdb_setrepcol authors, picture
```

This command returns replication information for the column picture in the table authors in the primary database.

Example 3

```
pdb_setrepcol authors, picture, enable
```

This command enables replication for the column picture in the table authors in the primary database.

Example 4

```
pdb_setrepcol all, disable
```

This command disables replication for all LOB columns in all marked tables in the primary database.

Usage

- When pdb_setrepcol is invoked, its function is determined by the keywords and options you specify.
- When multiple keywords and/or options are specified, each must be separated by a comma. Blank space before or after the comma(s) is optional. For example:

  ```
  pdb_setrepcol all, disable
  ```

- When you specify a column name in the pdb_setrepcol command, you must use the name of a valid LOB column.
- You cannot specify the following items as a table name in the pdb_setrepcol command:
  - Primary database system tables
  - Aliases or synonyms
  - Views
  - Replication Agent transaction log objects
- If a column name in the primary database is the same as a keyword, it can be identified by adding the string

  ```
  col=
  ```

  to the beginning of the column name. For example:

  ```
  pdb_setrepcol tablename, col=enable, disable
  ```

- If you enable LOB column replication with the pdb_setrepcol command, you should not configure the Replication Agent to convert date/time datatypes in the primary database. For more information, see "pdb_convert_datetime" on page 159.
- When pdb_setrepcol is invoked with either no argument or a single argument, it returns information about the enabled status of LOB columns in the primary database.
  - If pdb_setrepcol is invoked with no argument, it returns a list of all LOB columns for which replication is enabled in the primary database.

    Note Invoking the pdb_setrepcol command with no argument produces the same result as invoking the pdb_setrepcol command with the enable keyword.

  - If pdb_setrepcol is invoked with a table name, it returns information about the enabled status of all the LOB columns in the specified primary table.
  - If pdb_setrepcol is invoked with the enable keyword, it returns a list of all LOB columns for which replication is enabled in the primary database.
  - If pdb_setrepcol is invoked with the disable keyword, it returns a list of all LOB columns for which replication is disabled in the primary database.

  LOB columns for which replication is enabled are listed in the marked objects table.

  Note Any LOB columns in Replication Agent transaction log tables and shadow tables are not included in the list of LOB columns for which replication is disabled. Also not included are any synonyms, views, or aliases of these database objects.

  For LOB columns listed as disabled, transactions will not be captured for replication.
- When pdb_setrepcol is invoked with a valid primary table name and valid LOB column name, with no keywords, it returns information about the enabled status of the specified LOB column in the specified table in the primary database.

- When pdb_setrepcol is invoked with the all keyword, the operation specified by the following keyword (enable or disable) is applied to all LOB columns in marked tables in the primary database.
  - If pdb_setrepcol is invoked with the all keyword and the enable keyword, it enables replication for all LOB columns in marked tables in the primary database.
  - If pdb_setrepcol is invoked with the all keyword and the disable keyword, it disables replication for all LOB columns in marked tables in the primary database.
- When pdb_setrepcol is invoked with a valid primary table name and valid LOB column name followed by one or more keywords, the operation specified by the keyword (enable or disable) is applied to the specified LOB column in the specified primary table.
  - If pdb_setrepcol is invoked with a table name and LOB column name and the enable keyword, it enables replication for the specified LOB column in the primary database.
  - If pdb_setrepcol is invoked with a table name and LOB column name and the disable keyword, it disables replication for the specified LOB column in the primary database.

  If the table name and LOB column name combination you specify does not exist in the primary database, the pdb_setrepcol command returns an error.
- If the Replication Agent transaction log does not exist in the primary database, the pdb_setrepcol command returns an error.

See also

pdb_xlog, pdb_setrepproc, pdb_setreptable, ra_config

pdb_setrepproc

Description

Returns stored procedure replication marking status; unmarks all marked stored procedures or a specified stored procedure; enables or disables replication for all marked stored procedures or a specified stored procedure; marks a specified stored procedure for replication.

Note Sybase Replication Agent does not support procedure replication for DB2 Universal Database.

Syntax

To return stored procedure replication marking status:

pdb_setrepproc [*procname*|mark|unmark|enable|disable]

To unmark, enable or disable all marked stored procedures:

pdb_setrepproc all, {unmark[, force]|enable|disable}

To mark, unmark, enable, or disable a specified stored procedure:

pdb_setrepproc *procname*, {mark|unmark[, force]|enable|disable}

Parameters

procname

The name of the stored procedure in the primary database.

This parameter can be delimited with quote characters to specify the character case.

If mixed case (uppercase and lowercase) is required, the name must be delimited. For example:

```
"Proc"
```

Note If you must use an object name case that does not match the setting of the ltl_character_case parameter, the object name must be delimited.

If an object name contains any non-alphanumeric characters, such as spaces, periods, and so forth, it must be delimited with quote characters. For example:

```
"proc name"
"proc.name"
```

If an object name contains a period, it must be *both* owner-qualified and delimited with quote characters. For example:

```
owner."proc.name"
"proc.owner"."proc.name"
```

repname

The name of the stored procedure specified in the function replication definition for a primary stored procedure. This parameter can be delimited with quote characters to specify character case. See the previous description of the *procname* parameter for specifics.

By specifying a replicated name, stored procedure invocations can be replicated to a stored procedure invocation in the replicate database that has a different stored procedure name from the primary database.

Note The replicated name you specify with the pdb_setrepproc command must match a stored procedure name specified by a deliver as proc_name clause in a function replication definition for the primary database connection. The Replication Agent pdb_setrepproc command cannot verify the existence of a valid function replication definition, but if it does not exist, function replication from the primary table will fail.

all
: A keyword that refers to all marked stored procedures in the primary database. By using the all keyword, you can apply an unmark, enable, or disable operation to all marked stored procedures.

mark
: A keyword that refers to marking stored procedures for replication.

unmark
: A keyword that refers to unmarking marked stored procedures.

force
: A keyword that refers to the unmark operation.

  When the force keyword follows the unmark keyword, the pdb_setrepproc command immediately removes replication marking for the specified stored procedure in the primary database, without first checking the enable status of the stored procedure or checking for pending operations in the transaction log. When the force keyword follows the unmark keyword and the all keyword, the pdb_setrepproc command immediately removes replication marking from all marked stored procedures in the primary database, regardless of their enable status or any pending operations in the transaction log.

  The force keyword also forces complete execution of the unmarking script, even if errors occur during the unmarking process. Normally, when errors occur during script execution, the script terminates immediately without completing. The force keyword can be useful when a previous script execution failed and left the unmarking operation incomplete.

  When errors occur during a forced script execution, the pdb_setrepproc command returns the following message:

```
Errors were encountered and ignored during FORCEd script
execution. See error log for details.
```

enable
: A keyword that refers to enabling replication for marked stored procedures.

disable
A keyword that refers to disabling replication for marked stored procedures.

Examples

Example 1

```
pdb_setrepproc
```

This command returns replication marking information for all marked stored procedures in the primary database.

Example 2

```
pdb_setrepproc authors
```

This command returns replication marking information for the stored procedure named authors in the primary database.

Example 3

```
pdb_setrepproc authors, mark
```

This command marks the stored procedure named authors in the primary database.

Example 4

```
pdb_setrepproc authors, enable
```

This command enables replication for the stored procedure named authors in the primary database.

Example 5

```
pdb_setrepproc all, unmark
```

This command unmarks all marked stored procedures in the primary database.

Usage

- When pdb_setrepproc is invoked, its function is determined by the keywords and options you specify.
- When multiple keywords and/or options are specified, each must be separated by a comma. Blank space before or after the comma(s) is optional. For example:

  ```
  pdb_setrepproc all, unmark, force
  ```

- When you specify a stored procedure name in the pdb_setrepproc command, you must use the name of a valid stored procedure.
- You cannot specify the following items as a stored procedure name in the pdb_setrepproc command:
  - Primary database system procedures

  - Replication Agent transaction log procedures
- If a stored procedure name in the primary database is the same as a keyword, it can be identified by adding the string

  ```
  proc=
  ```

  to the beginning of the stored procedure name. For example:

  ```
  pdb_setrepproc proc=unmark, mark
  ```
- Marking and unmarking a stored procedure for replication requires that the Replication Agent drop then re-create the procedure. Ownership of the re-created procedure belongs to the user specified in the Replication Agent pds_username parameter. However, Replication Agent sets all the same privileges on the re-created procedure as were defined on the original procedure.

  Note Do *not* remove or alter the Replication Agent comments in a marked stored procedure.

- When pdb_setrepproc is invoked to mark a procedure for replication, Replication Agent does the following:
  - Modifies the user procedure to add code that captures input parameter values and generates Replication Agent transaction log records.
  - Generates a SQL script that creates the tables and procedures required for the Replication Agent transaction log in the primary database.
  - Saves the generated script in a file called *mark.sql* in the *rax-12_5\inst_name\scripts\procname* directory, where *inst_name* is the name of the Replication Agent instance and *procname* is the name of the stored procedure being marked.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the mark.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the procedure-marking scripts.

  - Executes the script to mark the stored procedure and create the transaction log objects in the primary database (if the value of the pdb_auto_run_scripts configuration parameter is true).
  - After the script completes successfully, moves the *mark.sql* file to the *rax-12_5\inst_name\scripts\procname\installed* directory.

- If the mark script fails, it is stored in a file (*mark.sql*) in the *rax-12_5\inst_name\scripts\procname* directory, the stored procedure is not marked, and transaction log objects are not created. You can examine the script by viewing the *mark.sql* file.
- When pdb_setrepproc is invoked to unmark a marked stored procedure, Replication Agent does the following:
  - Modifies the user procedure to remove Replication Agent code that captures input parameter values and generates transaction log records.
  - Generates a SQL script that removes the tables and procedures required for the transaction log in the primary database.
  - Saves the generated script in a file called *unmark.sql* in the *rax-12_5\inst_name\scripts\procname* directory, where *inst_name* is the name of the Replication Agent instance and *procname* is the name of the stored procedure being unmarked.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the unmark.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the procedure-unmarking scripts.

  - Executes the script to unmark the stored procedure and remove the transaction log objects in the primary database (if the value of the pdb_auto_run_scripts configuration parameter is true).
  - After the script completes successfully, moves the *unmark.sql* file to the *rax-12_5\inst_name\scripts\procname\installed* directory.
- If the unmark script fails, it is stored in a file (*unmark.sql*) in the *rax-12_5\inst_name\procname\scripts* directory and the stored procedure is not unmarked and the transaction log objects are not removed. You can examine the script by viewing the *unmark.sql* file. When the unmark script execution encounters a fatal error on any database object, the pdb_setrepproc command returns the following message:

  ```
  Could not unmark the following objects: ...
  See error log for details.
  ```

- When you use the unmark keyword to remove replication marking from a stored procedure, the pdb_setrepproc command verifies that replication is disabled for that stored procedure and checks to make sure that there are no pending (unprocessed) operations for that stored procedure in the

transaction log. If replication is not disabled or there is a pending operation for that stored procedure in the transaction log, pdb_setrepproc returns an error.

- When pdb_setrepproc is invoked with either no argument or a single argument, it returns marking information about the stored procedures in the primary database.
  - If pdb_setrepproc is invoked with no argument, it returns a list of all marked procedures in the primary database.

    Note Invoking the pdb_setrepproc command with no argument produces the same result as invoking the pdb_setrepproc command with the mark keyword.

  - If pdb_setrepproc is invoked with a procedure name, it returns complete marking information about the specified procedure.
  - If pdb_setrepproc is invoked with the mark keyword, it returns a list of all marked procedures in the primary database.
  - If pdb_setrepproc is invoked with the unmark keyword, it returns a list of all unmarked procedures in the primary database.
  - If pdb_setrepproc is invoked with the enable keyword, it returns a list of all marked procedures in the primary database for which replication is enabled.
  - If pdb_setrepproc is invoked with the disable keyword, it returns a list of all marked procedures in the primary database for which replication is disabled.

  Stored procedures marked for replication are listed in the marked objects table. All other user procedures are considered unmarked.

  Note The Replication Agent system procedures are not included in the list of unmarked procedures. Also not included are any synonyms or aliases of these procedures.

  For procedures listed as unmarked or disabled, their invocations will not be captured for replication.

- When pdb_setrepproc is invoked with the all keyword, the operation specified by the following keyword (unmark, enable, or disable) is applied to all marked procedures in the primary database.

- If pdb_setrepproc is invoked with the all keyword and the unmark keyword, it removes replication marking from all marked procedures in the primary database. You can also specify the force keyword after the unmark keyword to force immediate unmarking of all marked procedures, or to unmark procedures for which replication is still enabled or pending operations remain in the transaction log.
- If pdb_setrepproc is invoked with the all keyword and the enable keyword, it enables replication for all marked procedures in the primary database.
- If pdb_setrepproc is invoked with the all keyword and the disable keyword, it disables replication for all marked procedures in the primary database.

- When pdb_setrepproc is invoked with a valid procedure name followed by one or more keywords, the operation specified by the keyword (mark, unmark, enable, or disable) is applied to the specified procedure.
  - If pdb_setrepproc is invoked with a procedure name and the mark keyword, it marks the specified procedure in the primary database for replication.
  - If pdb_setrepproc is invoked with a procedure name and the unmark keyword, it removes replication marking from the specified procedure in the primary database. You can also specify the force keyword after the unmark keyword to force immediate unmarking of the specified procedure, to unmark a procedure for which replication is still enabled or pending operations remain in the transaction log, or to force the script execution to ignore errors and continue an unmarking operation that failed previously. If the unmark script execution encounters a fatal error on any database object, the pdb_setrepproc command returns the following message:

    ```
    Could not unmark the following objects: ...
    See error log for details.
    ```

  - If pdb_setrepproc is invoked with a procedure name and the enable keyword, it enables replication for the specified marked procedure in the primary database. If the enable script execution encounters a fatal error on any database object, the pdb_setrepproc command returns the following message:

    ```
    Could not enable the following objects: ...
    See error log for details.
    ```

- If pdb_setrepproc is invoked with a procedure name and the disable keyword, it disables replication for the specified marked procedure in the primary database. If the disable script execution encounters a fatal error on any database object, the pdb_setrepproc command returns the following message:

  ```
  Could not disable the following objects: ...
  See error log for details.
  ```

  If the procedure name you specify does not exist in the primary database, the pdb_setrepproc command returns an error.

- When pdb_setrepproc is invoked with a procedure name and a replicated name, followed by the mark keyword, the primary procedure is marked for replication with the specified replicated name.

  If the primary procedure name you specify does not exist in the primary database, the pdb_setrepproc command returns an error.

  By specifying a replicated name, procedure invocations can be replicated to a procedure in the replicate database that has a different name from the primary procedure.

  Note The replicated name you specify with the pdb_setrepproc command must match a procedure name specified by a deliver as clause in a function replication definition for the primary database connection. The Replication Agent pdb_setrepproc command cannot verify the existence of a valid replication definition, but if it does not exist, replication of the primary procedure invocation will fail.

- If the Replication Agent transaction log does not exist in the primary database, the pdb_setrepproc command returns an error.

See also pdb_xlog, pdb_setrepcol, pdb_setreptable, ra_config

pdb_setreptable

Description Returns replication marking status; unmarks all marked tables or a specified table; enables or disables replication for all marked tables or a specified table; marks a specified table for replication.

Syntax To return replication marking status:

pdb_setreptable [*tablename*|mark|unmark|enable|disable]

To unmark, enable or disable all tables:

```
pdb_setreptable all, {unmark[, force]|enable|disable}
```

To mark, unmark, enable, or disable a specified table:

```
pdb_setreptable tablename, {mark[, owner]|
 unmark[, force] |enable|disable}
```

To mark a specified table for replication with a replicated name:

```
pdb_setreptable tablename, repname, mark[, owner]
```

Parameters

tablename

The name of a user table in the primary database.

The *tablename* parameter can be owner-qualified to include the primary table owner name, with each element separated by a period. For example:

```
owner.table
```

Note If you want to use owner-qualified table names for either primary tables or replicate tables, you must set the value of the Replication Agent use_rssd parameter to true.

This parameter can be delimited with quote characters to specify the character case.

If mixed case (uppercase and lowercase) is required, the name must be delimited. For example:

```
"Owner".table
"Owner"."Table"
```

Each mixed case element of the *tablename* parameter must be delimited separately, as illustrated in the previous example.

Note If you must use an object name case that does not match the value of the ltl_character_case parameter, the object name must be delimited.

If an object name contains any non-alphanumeric characters, such as spaces, periods, and so forth, it must be delimited with quote characters. For example:

```
"table name"
owner."table name"
```

If an object name contains a period, it must be *both* owner-qualified and delimited with quote characters. For example:

```
owner."table.name"
"table.owner"."table.name"
```

repname
The name of the table specified in the replication definition for a primary table.

Note The replicated name (including the owner name) you specify with the pdb_setreptable command must match a table name specified by a with all tables named clause in a replication definition for the primary database connection. The Replication Agent pdb_setreptable command cannot verify the existence of a valid replication definition, but if it does not exist, replication from the primary table will fail.

The *repname* parameter can be owner-qualified to include the replicate table owner name, with each element separated by a period. For example:

```
repowner.reptable
```

Note If you want to use an owner-qualified replicate table name with the replicate owner's name, do *not* use the owner keyword with the pdb_setreptable command. If you use the owner keyword and specify a replicate table name, the primary table owner name is sent with the replicate table name in the LTL.

This parameter can also be delimited with quote characters to specify the character case. See the previous description of the *tablename* parameter for specifics.

all
A keyword that refers to all marked tables in the primary database. By using the all keyword, you can apply an unmark, enable, or disable operation to all marked tables.

mark
A keyword that refers to replication marking.

When the mark keyword is the only option, the pdb_setreptable command returns a list of all marked tables in the primary database. When the mark keyword follows a table name (and optionally a replicated name), the pdb_setreptable command marks the specified table in the primary database.

owner
A keyword that refers to the mark operation.

When the optional owner keyword follows the mark keyword, the pdb_setreptable command marks the specified table in the primary database so that when operations against that table are replicated, the primary table owner name is sent along with the table name in the form owner.tablename when LTL is sent to the primary Replication Server.

Note If you want to use owner-qualified table names for either primary tables or replicate tables, you must set the value of the Replication Agent use_rssd parameter to true.

Note If you want to use an owner-qualified replicate table name with the replicate owner's name, do *not* use the owner keyword with the pdb_setreptable command. If you use the owner keyword and specify a replicate table name, the primary table owner name is sent with the replicate table name in the LTL.

---

unmark

A keyword that refers to unmarking a marked table.

When the unmark keyword is the only option, the pdb_setreptable command returns a list of all tables in the primary database that are not marked for replication. When the unmark keyword follows a table name, the pdb_setreptable command removes replication marking for the specified table in the primary database. When the unmark keyword follows the all keyword, the pdb_setreptable command removes replication marking for all marked tables in the primary database.

force

A keyword that refers to the unmark operation.

When the force keyword follows the unmark keyword, the pdb_setreptable command immediately removes replication marking for the specified table in the primary database, without first checking the enable status of the table or checking for pending operations in the transaction log. When the force keyword follows the unmark keyword and the all keyword, the pdb_setreptable command immediately removes replication marking from all marked tables in the primary database, regardless of their enable status or any pending operations in the transaction log.

The force keyword also forces complete execution of the unmarking script, even if errors occur during the unmarking process. Normally, when errors occur during script execution, the script terminates immediately without

completing. The force keyword can be useful when a previous script execution failed and left the unmarking operation incomplete.

When errors occur during a forced script execution, the pdb_setreptable command returns the following message:

```
Errors were encountered and ignored during FORCEd script
execution. See error log for details.
```

enable
: A keyword that refers to enabling replication for marked tables.

  When the enable keyword is the only option, the pdb_setreptable command returns a list of all marked tables in the primary database for which replication is enabled. When the enable keyword follows a table name, the pdb_setreptable command enables replication for the specified table in the primary database. When the enable keyword follows the all keyword, the pdb_setreptable command enables replication for all marked tables in the primary database.

disable
: A keyword that refers to disabling replication for marked tables.

  When the disable keyword is the only option, the pdb_setreptable command returns a list of all marked tables in the primary database for which replication is disabled. When the disable keyword follows a table name, the pdb_setreptable command disables replication for the specified table in the primary database. When the disable keyword follows the all keyword, the pdb_setreptable command disables replication for all marked tables in the primary database.

Examples

Example 1

```
pdb_setreptable authors
```

This command returns replication marking information for the table named authors in the primary database.

Example 2

```
pdb_setreptable mark
```

This command returns replication marking information for all marked tables in the primary database.

Example 3

```
pdb_setreptable disable
```

This command returns replication marking information for all marked tables for which replication has been disabled in the primary database.

Example 4

```
pdb_setreptable all, unmark, force
```

This command forces unmarking for all marked tables in the primary database.

Example 5

```
pdb_setreptable all, enable
```

This command enables replication for all marked tables in the primary database.

Example 6

```
pdb_setreptable authors, mark
```

This command marks for replication the table named authors in the primary database.

Example 7

```
pdb_setreptable authors, mark, owner
```

This command marks for replication the table named authors in the primary database so that the name of the primary table owner will be passed along with the table name in the LTL.

Example 8

```
pdb_setreptable authors, auth_name, mark
```

This command marks for replication the table named authors in the primary database with a replicate name auth_name.

Example 9

```
pdb_setreptable authors, auth_name, mark, owner
```

This command marks for replication the table named authors in the primary database with a replicate name auth_name so that the name of the primary table owner will be passed along with the replicate name in the LTL.

Example 10

```
pdb_setreptable authors, bob.auth_name, mark
```

This command marks for replication the table named authors in the primary database with a replicate name auth_name so that the name of the replicate table owner (bob) will be passed along with the replicate name in the LTL.

Example 11

```
pdb_setreptable authors, enable
```

This command enables replication for the marked table authors in the primary database.

Example 12

```
pdb_setreptable table=mark, enable
```

This command enables replication for the marked table named mark in the primary database.

Example 13

```
pdb_setreptable authors, unmark, force
```

This command forces unmarking for the marked table authors in the primary database.

Usage

- When pdb_setreptable is invoked, its function is determined by the keywords and options you specify.
- When multiple keywords and/or options are specified, each must be separated by a comma. Blank space before or after the comma(s) is optional. For example:

  ```
  pdb_setreptable all, unmark, force
  ```

- When you specify a primary table in the pdb_setreptable command, you must use the name of a valid primary table.
- You cannot specify the following items as a primary table in the pdb_setreptable command:
  - Primary database system tables
  - Views
  - Replication Agent transaction log tables
- If you specify an alias or synonym as a primary table in the pdb_setreptable command, the actual table that the alias or synonym refers

to is acted upon. The actual table name is the table name sent to the primary Replication Server.

- If a table name in the primary database is the same as a keyword, it can be identified by adding the string

  ```
  table=
  ```

  to the beginning of the table name. For example:

  ```
  pdb_setreptable table=unmark, mark
  ```

  This is true for both primary table names and replicated names.

- When pdb_setreptable is invoked to mark a table for replication, trigger-based Replication Agents (for Informix, Microsoft SQL Server, and Oracle) do the following:

  - Generate a SQL script that creates the tables, procedures, and triggers required for the transaction log in the primary database.
  - Save the generated script in a file called *mark.sql* in the *rax-12_5\inst_name\scripts\tablename* directory, where *inst_name* is the name of the Replication Agent instance and *tablename* is the name of the table being marked.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the mark.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the table-marking scripts.

  - Execute the script to mark the table and create the transaction log objects in the primary database (if the value of pdb_auto_run_scripts is true).
  - After the script completes successfully, move the *mark.sql* file to the *rax-12_5\inst_name\scripts\tablename\installed* directory.
  - If a user trigger exists on the table, modify the user trigger to add code that captures data and generates Replication Agent transaction log records.

- If the mark script fails, it is stored in a file (*mark.sql*) in the *rax-12_5\inst_name\scripts\tablename* directory, the table is not marked, and transaction log objects are not created. You can examine the script by viewing the *mark.sql* file.

- When pdb_setreptable is invoked to unmark a marked primary table, trigger-based Replication Agents (for Informix, Microsoft SQL Server, and Oracle) do the following:
  - Generate a SQL script that removes the tables, procedures, and triggers required for the transaction log in the primary database.
  - Save the generated script in a file called *unmark.sql* in the *rax-12_5\inst_name\scripts\tablename* directory, where *inst_name* is the name of the Replication Agent instance and *tablename* is the name of the table being unmarked.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the unmark.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the table-unmarking scripts.

  - Execute the script to unmark the table and remove the transaction log objects in the primary database (if the value of pdb_auto_run_scripts is true).
  - After the script completes successfully, move the *unmark.sql* file to the *rax-12_5\inst_name\scripts\tablename\installed* directory.
  - If a user trigger existed on the table when it was marked, modify the user trigger to remove Replication Agent code that captures data and generates transaction log records.
- If the unmark script fails, it is stored in a file (*unmark.sql*) in the *rax-12_5\inst_name\tablename\scripts* directory, the table is not unmarked, and the transaction log objects are not removed. You can examine the script by viewing the *unmark.sql* file. When the unmark script execution encounters a fatal error on any database object, the pdb_setreptable command returns the following message:

  ```
  Could not unmark the following objects: ...
  See error log for details.
  ```

- When you use the unmark keyword to remove replication marking from a primary table, the pdb_setreptable command verifies that replication is disabled for that table and checks to make sure that there are no pending (unprocessed) operations for that table in the transaction log. If replication is not disabled or there is a pending operation for that table in the transaction log, pdb_setreptable returns an error.

- When you use the unmark keyword to remove replication marking from primary tables, you can also specify the force keyword to immediately remove replication marking from primary tables, without regard to whether replication is disabled or whether pending operations exist in the transaction log. The force keyword also ignores script execution errors. If the unmark script execution encounters a fatal error on any database object, the pdb_setreptable command returns the following message:

  ```
  Could not unmark the following objects: ...
  See error log for details.
  ```

- When pdb_setreptable is invoked with either no argument or a single argument, it returns marking information about the tables in the primary database.

  - If pdb_setreptable is invoked with no argument, it returns a list of all marked tables in the primary database.

    Note Invoking the pdb_setreptable command with no argument produces the same result as invoking the pdb_setreptable command with the mark keyword.

  - If pdb_setreptable is invoked with a table name, it returns complete marking information about the specified primary table.
  - If pdb_setreptable is invoked with the mark keyword, it returns a list of all marked tables in the primary database.
  - If pdb_setreptable is invoked with the unmark keyword, it returns a list of all unmarked tables in the primary database.
  - If pdb_setreptable is invoked with the enable keyword, it returns a list of all marked tables in the primary database for which replication is enabled.
  - If pdb_setreptable is invoked with the disable keyword, it returns a list of all marked tables in the primary database for which replication is disabled.

  Tables marked for replication are listed in the marked objects table. All other user tables are considered unmarked.

Note The Replication Agent transaction log tables and shadow tables are not included in the list of unmarked tables. Also not included are any synonyms, views, or aliases of these database objects.

For tables listed as unmarked or disabled, transactions will not be captured for replication.

- When pdb_setreptable is invoked with the all keyword, the operation specified by the following keyword (unmark, enable, or disable) is applied to all marked tables in the primary database.
  - If pdb_setreptable is invoked with the all keyword and the unmark keyword, it removes replication marking from all marked tables in the primary database. You can also specify the force keyword after the unmark keyword to force immediate unmarking of all marked tables, or to unmark tables for which replication is still enabled or pending operations remain in the transaction log.
  - If pdb_setreptable is invoked with the all keyword and the enable keyword, it enables replication for all marked tables in the primary database.
  - If pdb_setreptable is invoked with the all keyword and the disable keyword, it disables replication for all marked tables in the primary database.
- When pdb_setreptable is invoked with a valid primary table name followed by one or more keywords, the operation specified by the keyword (mark, unmark, enable, or disable) is applied to the primary table.
  - If pdb_setreptable is invoked with a table name and the mark keyword, it marks the specified table in the primary database for replication. You can also specify the owner keyword after the mark keyword so that when operations against the table are replicated, the primary table owner name will be attached to the table name in the form owner.tablename.

    Note If you want to use owner-qualified table names for either primary tables or replicate tables, you must set the value of the Replication Agent use_rssd parameter to true.

  - If pdb_setreptable is invoked with a table name and the unmark keyword, it removes replication marking from the specified table in the primary database. You can also specify the force keyword after the unmark keyword to force immediate unmarking of the specified table, to unmark a table for which replication is still enabled or pending operations remain in the transaction log, or to force the script execution to ignore errors and continue an unmarking operation that failed previously.

- If pdb_setreptable is invoked with a table name and the enable keyword, it enables replication for the specified marked table in the primary database. If the enable script execution encounters a fatal error on any database object, the pdb_setreptable command returns the following message:

  ```
  Could not enable the following objects: ...
  See error log for details.
  ```

- If pdb_setreptable is invoked with a table name and the disable keyword, it disables replication for the specified marked table in the primary database. If the disable script execution encounters a fatal error on any database object, the pdb_setreptable command returns the following message:

  ```
  Could not disable the following objects: ...
  See error log for details.
  ```

If the table name you specify does not exist in the primary database, the pdb_setreptable command returns an error.

- When pdb_setreptable is invoked with a primary table name and a replicated name, followed by the mark keyword, the primary table is marked for replication with the specified replicated name.

  If the primary table name you specify does not exist in the primary database, the pdb_setreptable command returns an error.

  By specifying a replicated name, transactions can be replicated to a table in the replicate database that has a different name from the primary table.

  Note The replicated name you specify with the pdb_setreptable command must match a table name specified by a with all tables named clause in a replication definition for the primary database connection. The Replication Agent pdb_setreptable command cannot verify the existence of a valid replication definition, but if it does not exist, replication from the primary table will fail.

  You can also specify the owner keyword after the mark keyword so that when operations against the primary table are replicated, the primary table owner name will be attached to the replicate table name in the form owner.tablename.

Note If you want to use an owner-qualified replicate table name with the replicate owner's name, do *not* use the owner keyword with the pdb_setreptable command. If you use the owner keyword and specify a replicate table name, the primary table owner name is sent with the replicate table name in the LTL.

- If the Replication Agent transaction log does not exist in the primary database, the pdb_setreptable command returns an error.

See also pdb_xlog, pdb_setrepcol, pdb_setrepproc, ra_config

pdb_truncate_xlog

Description Truncates the Replication Agent transaction log on demand.

Note Sybase Replication Agent does not support log truncation for DB2 Universal Database.

Syntax pdb_truncate_xlog

Usage

- When pdb_truncate_xlog is invoked, Replication Agent immediately truncates the transaction log based on the most recent truncation point received from the primary Replication Server. The truncation point is part of the information contained in the LTM Locator.
- To update the LTM Locator from the primary Replication Server, use the ra_locator command.
- The pdb_truncate_xlog command is asynchronous and it does not return success or failure (unless an immediate error occurs). You must examine the Replication Agent system log to determine success or failure of the pdb_truncate_xlog command.
- You can use the ra_statistics command to view the "Number of transactions truncated," both before and after you use the pdb_truncate_xlog command.
- If the Replication Agent transaction log does not exist or if a connection failure occurs, the pdb_truncate_xlog command returns an error message.
- You can use the ra_config command to specify the type of automatic truncation you want. You can use the pdb_truncate_xlog command to

truncate the transaction log if automatic truncation is not sufficient to manage the size of the transaction log.

- The pdb_truncate_xlog command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_xlog, ra_config, ra_locator

pdb_version

Description Returns the type and version of the primary database server and JDBC driver.

Syntax pdb_version

Examples

```
pdb_version
```

This command returns the primary data server vendor name and version and the JDBC driver name and version for the primary database.

Usage

- When pdb_version is invoked, it returns the product name and version information for the primary database server and JDBC driver.

  Note When Replication Agent is used with a database gateway connection to the primary database server, version information about the database gateway software is returned by the pdb_version command, instead of information about the primary database server.

- Actual results returned vary depending on the type of primary database server.
- If the primary database connection is down or not configured, the pdb_version command returns an error message.
- The pdb_version command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_xlog, ra_version, ra_version_all

pdb_xlog

Description Returns the names of transaction log base objects, creates transaction log base objects in the primary database, or removes transaction log base objects from the primary database.

Syntax

pdb_xlog [create|remove [, force]]

Parameters

create
The keyword for creating a transaction log.

remove
The keyword for removing a transaction log.

force
A keyword that refers to the remove operation.

Usage

- When pdb_xlog is invoked with no argument, it returns the actual names (not synonyms or aliases) of all Replication Agent transaction log base objects in the primary database.

  Note See the appendix for your specific primary database server at the end of this book for more information on transaction log object names.

- If pdb_xlog is invoked with no argument and the Replication Agent transaction log base objects do not exist in the primary database, the pdb_xlog command returns no information.
- When pdb_xlog is invoked with the create keyword, Replication Agent does the following:
  - Generates a SQL script that creates the tables and procedures required for the transaction log base objects in the primary database.
  - Saves the generated script in a file called *create.sql* in the *rax-12_5\inst_name\scripts\xlog* directory, where *inst_name* is the name of the Replication Agent instance.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the create.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the transaction log creation scripts.

  - Executes the script to create the Replication Agent transaction log base objects in the primary database (if the value of the pdb_auto_run_scripts configuration parameter is true).
  - After the script completes successfully, moves the *create.sql* file to the *rax-12_5\inst_name\scripts\xlog\installed* directory.

- If the create script fails, it is stored in a file (*create.sql*) in the *rax-12_5\inst_name\scripts\xlog* directory and the transaction log is not created. You can examine the script by viewing the *create.sql* file.
- If pdb_xlog is invoked with the create keyword and transaction log base objects already exist in the primary database (using the prefix string specified by the pdb_xlog_prefix configuration parameter), then pdb_xlog returns an error message.
- When pdb_xlog is invoked with the remove keyword, Replication Agent does the following:
  - Generates a SQL script that deletes the tables and procedures required for the transaction log base objects in the primary database.
  - Saves the generated script in a file called *remove.sql* in the *rax-12_5\inst_name\scripts\xlog* directory, where *inst_name* is the name of the Replication Agent instance.

    Note If the value of the pdb_auto_run_scripts configuration parameter is false, the remove.sql script will be saved but not executed automatically. When the value of the pdb_auto_run_scripts configuration parameter is false, you must manually execute the transaction log removal scripts.

  - Executes the script to delete the transaction log base objects in the primary database (if the value of the pdb_auto_run_scripts configuration parameter is true).
  - After the script completes successfully, moves the *remove.sql* file to the *rax-12_5\inst_name\scripts\xlog\installed* directory.
- If the remove script fails, it is stored in a file (*remove.sql*) in the *rax-12_5\inst_name\scripts\xlog* directory and the transaction log is not deleted. You can examine the script by viewing the *remove.sql* file.
- When pdb_xlog is invoked with the remove keyword followed by the force keyword, the remove.sql script continues executing, even if errors occur. The force keyword may be useful when a previous remove operation failed and the remove.sql script terminated with an error.
- If pdb_xlog is invoked with the remove keyword and transaction log base objects do not exist in the primary database (using the prefix string specified by the pdb_xlog_prefix configuration parameter), then pdb_xlog returns an error message.

- If pdb_xlog is invoked with the remove keyword and any objects in the primary database are still marked for replication, then pdb_xlog returns an error message.

  You can use the pdb_setrepproc and pdb_setreptable commands to determine which stored procedures and tables in the primary database are still marked.

  Even if objects are marked in the primary database, you can use the pdb_xlog command with the remove keyword followed by the force keyword to unmark any marked objects and then remove the transaction log objects.
- If pdb_xlog is invoked with no argument, the command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.
- If pdb_xlog is invoked with either the create or remove keyword, the command is valid only when the Replication Agent instance is in the *Admin* state.
- For more detailed information about the Replication Agent transaction log components and structure, see the appendix for your specific primary database server at the end of this book.

See also pdb_setrepcol, pdb_setrepproc, pdb_setreptable, ra_config

quiesce

Description Stops all Replication Agent processing in the *Replicating* state and puts the Replication Agent instance in the *Admin* state.

Syntax quiesce

Usage
- When the quiesce command is invoked, it stops all current Replication Agent processing.
  - The Log Reader component stops reading operations from the transaction log as soon as the current scan is complete.
  - The Log Transfer Interface (LTI) component stops sending LTL to the primary Replication Server as soon as its input queue is empty.
  - When the LTI component is finished processing its input queue and sending the resulting LTL, the Replication Agent instance drops all its connections to the primary database and the primary Replication Server.

  - The Replication Agent instance goes from the *Replicating* state to the *Admin* state.
- If the Replication Agent internal queues are filled with large amounts of data when the quiesce command is invoked, the quiesce processing may take a while to complete.
- If the Replication Agent instance is in the *Admin* state, the quiesce command returns an error message.
- The quiesce command is valid only when the Replication Agent instance is in the *Replicating* state.

Note The action of the suspend command is similar to that of the quiesce command, except that the suspend command stops Replication Agent processing immediately and flushes any pending change data in the Replication Agent internal queues.

See also ra_status, resume, shutdown, suspend

ra_config

Description Returns help information for Replication Agent configuration parameters, or sets the value of a specified Replication Agent configuration parameter.

Syntax ra_config [*param*[, *value*]]

Parameters

param
The name of a Replication Agent configuration parameter.

value
The value to be assigned to the configuration parameter specified in the *param* option. You can use the keyword default to set the specified parameter to its default value.

Examples

Example 1

```
ra_config pdb_convert_datetime
```

This command returns the current value of the pdb_convert_datetime configuration parameter.

Example 2

```
ra_config scan_sleep_max, 60
```

This command changes the value of the scan_sleep_max parameter to 60.

Usage

- If ra_config is invoked with no options specified, it returns a list of all Replication Agent configuration parameters. The following information is returned for each configuration parameter:
  - parameter name
  - parameter type – This is actually the datatype of the parameter's value. For example, string, numeric, or boolean.
  - current value – The value of the parameter in effect at the time the ra_config command is invoked.
  - pending value – If different from the current value, this is the value to which the parameter was set by a previous invocation of the ra_config command, but which has not yet taken effect.
  - default value – The value of the parameter when the Replication Agent configuration file is created.
  - legal values – The values that are allowed for the parameter, for example, a range of numbers or a list of specific strings.
  - category – This refers to the Replication Agent component affected by the value of the parameter.
  - restart – This refers to parameters that require the Replication Agent instance to be shut down and restarted before a change in value takes effect.
- If ra_config is invoked with only a *param* option specified, it returns information only for the specified configuration parameter.
- If ra_config is invoked with both *param* and *value* options specified, it changes the setting of the specified configuration parameter to the value specified in the *value* option.
- You can use the keyword default in place of the *value* option to reset a configuration parameter to its default value. For example:

  ```
  ra_config use_rssd, default
  ```

- When ra_config is invoked with no options or only the *param* option specified, the command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.
- If ra_config is invoked in the *Replicating* state, with both the *param* and *value* options specified for a parameter that can be changed only in the *Admin* state, it returns an error message.

- When ra_config is invoked with both the *param* and *value* options specified, the command is always valid when the Replication Agent instance is in the *Admin* state.
- For information on configuration parameters, see Chapter 5, "Sybase Replication Agent Configuration Parameters".

See also pdb_setrepcol, pdb_setrepproc, pdb_setreptable, pdb_xlog, ra_help, ra_set_login

ra_date

Description Returns the current date and time from the Replication Agent server.

Syntax ra_date

Examples

```
ra_date
```

This command returns the current date and time from the Replication Agent server, as shown below:

```
Current RA Date
 ----------------------------
     Nov 30 2000 12:09:47.310
(1 row affected)
```

Usage
- When ra_date is invoked, it returns the current date and time from the Replication Agent server in the form of a Sybase datetime datatype.
- The ra_date command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also ra_config, pdb_date

ra_dump

Description Emulates the Replication Server rs_dumpdb and rs_dumptran system functions.

Syntax ra_dump [database|transaction,] *dbname*, *dump_label*

Parameters

database
A keyword that causes the primary Replication Server to apply the function string associated with the rs_dumpdb system function.

transaction
A keyword that causes the primary Replication Server to apply the function string associated with the rs_dumptran system function.

dbname
The name of the database to be dumped.

dump_label
A varchar(30) value that contains information to identify the dump.

Usage

- When ra_dump is invoked, Replication Agent places a "dump" marker in the Replication Agent transaction log to facilitate coordinated dumps.
- The ra_dump command returns an error message if the transaction log does not exist.
- The ra_dump command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.
- For more information about the Replication Server rs_dumpdb and rs_dumptran system functions, refer to the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

See also ra_config, ra_marker, ra_statistics

ra_help

Description Returns help information for Replication Agent commands.

Syntax ra_help [*command*]

Parameters *command*
The name of a Replication Agent command for which you want to view help information.

Examples **Example 1**

```
ra_help
```

This command returns help for all Replication Agent commands.

Example 2

```
ra_help pdb_xlog
```

This command returns help for the pdb_xlog command.

Usage

- If ra_help is invoked with no option specified, it returns help information for all Replication Agent commands.
- If ra_help is invoked with the *command* option specified, it returns help information only for the specified Replication Agent command.

- The ra_help command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also ra_config

ra_locator

Description Returns the current value of the LTM locator, retrieves an LTM locator value from Replication Server, or sets the value of the LTM locator to all zeros.

Syntax ra_locator [update|zero]

Parameters

update
The optional keyword that tells Replication Agent to retrieve a new LTM locator from the primary Replication Server.

zero
The optional keyword that tells Replication Agent to set the value of the LTM locator stored in the Replication Agent transaction log to all zeros.

Examples

Example 1

```
ra_locator update
```

This command tells Replication Agent to retrieve a new LTM locator value from the primary Replication Server.

Example 2

```
ra_locator zero
```

This command sets the value of the LTM locator maintained by Replication Agent to all zeros.

Example 3

```
ra_locator
```

This command returns the current value of the LTM locator maintained by Replication Agent, as shown below:

```
Locator
 --------------------------------------------------
 0000000052000000000000527FFFFFFFFFFFFFF0022FB3B
 (1 row affected)
```

Usage

- When ra_locator is invoked with no keyword, it returns the current value of the LTM locator maintained by the Replication Agent instance. The

Replication Agent stores the value of the LTM locator in a table in the primary database.

Note The value of the LTM locator maintained by the Replication Agent is known as the *origin queue ID*.

- When ra_locator is invoked with the update keyword, it retrieves a new LTM locator value from the primary Replication Server and stores it in the transaction log system table. The update operation is asynchronous and may take a few seconds to complete.

Note When the ra_locator command is invoked with the update keyword, the change takes effect only if the Replication Agent instance is in the *Replicating* state.

- When ra_locator is invoked with the zero keyword, it sets the value of the LTM locator maintained by Replication Agent to all zeros.
- The LTM locator contains information used by Replication Agent to determine where to start reading the transaction log. Upon start-up or connection failure recovery, Replication Agent requests an LTM locator from the primary Replication Server. If the value of the LTM locator returned from the primary Replication Server is all zeros, then Replication Agent uses the value of the LTM locator it has stored. If the value of the LTM locator that Replication Agent has stored is all zeros, then the Replication Agent Log Reader component starts reading the transaction log from either the current beginning of the log (trigger-based Replication Agents), or from the end of the log (log-based Replication Agents).
- For more information about the format of the origin queue ID, see the appendix for your specific primary database server at the end of this book.
- If the Replication Agent transaction log does not exist, the ra_locator command returns an error message.
- The ra_locator command with the zero keyword is valid only when the Replication Agent instance is in the *Admin* state.

See also pdb_gen_id, pdb_truncate_xlog

ra_maintid

Description

Returns the name of the Replication Server maintenance user ID for the primary database connection.

Syntax

ra_maintid

Examples

```
ra_maintid
```

This command returns the Replication Server maintenance user ID currently stored in the Replication Agent transaction log, as shown below:

```
Maintenance User
 ----------------
 SYS
(1 row affected)
```

Usage

- Replication Server requires a maintenance user ID for each database connection. Each maintenance user ID is associated with a specific Replication Server database connection. The maintenance user ID for each database connection is defined with the Replication Server create connection command.
- The filter_maint_userid configuration parameter is provided to support bidirectional replication, wherein the primary database is also a replicate database that has transactions applied to it by a Replication Server.

  If the value of the filter_maint_userid parameter is true, database operations applied by the Replication Server maintenance user ID are *not* replicated. The Log Reader component filters out the Replication Server maintenance user ID when it reads the transaction log.
- If ra_maintid is invoked when the primary database connection is down, it returns an error message.
- The ra_maintid command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

Note Each time the Replication Agent goes to *Replicating* state, it retrieves the maintenance user ID from the primary Replication Server. If you invoke the ra_maintid command when the Replication Agent is in the *Admin* state, the value retrieved from the transaction log may not be correct. The correct maintenance user ID is always returned if you invoke the ra_maintid command when the Replication Agent is in the *Replicating* state.

See also

ra_config, ra_statistics

ra_marker

Description — Emulates the Replication Server rs_marker system function.

Syntax — ra_marker *command_tag*

Parameters

command_tag
A varchar(255) value that contains information used for subscription materialization.

Examples

```
ra_marker 'activate subscription 309 0 with suspension'
```

This command invokes the Replication Server activate subscription command.

Usage

- When ra_marker is invoked, Replication Agent inserts a row in a shadow table marked for replication. The replicated transaction sends a marker object to the primary Replication Server in the LTL output from the Replication Agent.
- The ra_marker command returns an error message if the transaction log does not exist.
- The ra_marker command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.
- For more information about the Replication Server rs_marker system function, refer to the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

See also — ra_config, ra_dump, ra_statistics

ra_set_login

Description — Sets the Replication Agent administrative user login and password.

Syntax — ra_set_login *username*, *password*

Parameters

username
The login name (user ID) of the administrative user for the Replication Agent instance.

password
The password of the administrative user for the Replication Agent instance.

Examples

```
ra_set_login Bob3, bug3wag
```

This command sets the Replication Agent administrative user ID to “Bob3” and the password for that user to “bug3wag.”

Usage

- The Replication Agent administrative user is the user ID with permission to log in to the Replication Agent instance through the administration port.
- Exactly one Replication Agent administrative user ID is valid at any time.
- Any change in the Replication Agent administrative user ID or password takes place immediately.
- The password specified for the administrative user ID is encrypted in the configuration file.
- The ra_set_login command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also

ra_config

ra_statistics

Description

Returns performance-related statistics for Replication Agent components or resets statistics counters.

Syntax

ra_statistics [*component*|reset]

Parameters

component

The optional keyword that identifies a Replication Agent component. Valid *component* keywords are:

LR – Log Reader component

LTI – Log Transfer Interface component

LTM – Log Transfer Manager component

reset

The optional keyword that resets statistics counters.

Examples

Example 1

```
ra_statistics
```

This command returns performance statistics for the Replication Agent instance.

Example 2

```
ra_statistics reset
```

This command resets the statistics counters for the Replication Agent instance.

Usage

- If you invoke ra_statistics without specifying a *component* option, it returns statistics for all Replication Agent components.
- If you invoke ra_statistics and specify a *component* option, it returns statistics for the Log Transfer Manager, as well as the component you specify.
- Statistics counters are reset automatically each time the Replication Agent instance goes into the *Replicating* state.
- If you invoke ra_statistics with the reset keyword, Replication Agent immediately resets all statistics, except the following:
  - Time statistics obtained
  - Time replication last started
  - Time statistics last reset
  - VM total memory
  - VM free memory

    Note The "VM total memory" and "VM free memory" statistics values are refreshed each time statistics are returned.

  - Last QID sent
  - Last transaction ID sent
- All statistics that give a measurement of time are in seconds.
- Table 4-2 lists the statistics that are returned by the ra_statistics command.

Table 4-2: Replication Agent statistics

Component	Statistic	Description
LTM	Time statistics obtained	Day, date, and time ra_statistics was invoked and information returned.
LTM	Time replication last started	Day, Date, and time that the *Replicating* state was entered.
LTM	Time statistics last reset	Day, date, and time statistics counters were reset.
LTM	VM total memory	Total memory (in bytes) allocated to the Java Virtual Machine on which Replication Agent is running.
LTM	VM free memory	Total memory (in bytes) allocated but not used by the Java Virtual Machine on which Replication Agent is running.

Component	Statistic	Description
LR	Number of XLog scans	Number of times Log Reader scanned transaction log since last reset.
LR	Avg unprocessed operations per XLog scan	Average number of operations not processed for any reason during each log scan.
LR	Avg XLog scan time	Average time required for transaction log scans since last reset.
LR	Number of operations replicated	Number of operations read from transaction log and passed to LTI since last reset.
LR	Number of transactions replicated	Total number of transactions replicated since last reset.
LR	Number of XLog operations skipped (maint_user, unmarked tables)	Total number of maintenance user transactions in primary database skipped and operations with entries pointing to non-existent shadow tables skipped since last reset.
LR	Avg wait time on empty XLog	Average time Log Reader waits for operations to be recorded in the transaction log.
LR	Avg PDB Service Time / Operation	Average Log Reader processing time per replicated operation.
LR	Operation Queue Size	Number of metadata objects in the Log Reader internal queue.
LR	Operation Data Hash Size	Number of data objects in the Log Reader internal data hash table.
LR	Number of transactions truncated	Total number of transactions truncated from transaction log since last reset.
LTI	Total bytes sent	Total number of bytes sent to Replication Server since last reset.
LTI	Number of LTL commands sent	Total number of LTL commands sent to Replication Server since last reset.
LTI	Avg LTL commands/sec	Average number of LTL commands sent to Replication Server per second.
LTI	Avg LTL command size	Average size of the LTL commands sent since last reset.
LTI	Avg Rep Server turnaround time	Average time it takes Replication Server to acknowledge each LTL buffer sent.
LTI	Avg Bytes/second during transmission	Average bytes per second rate over connection to Replication Server.

Component	Statistic	Description
LTI	Avg data arrival time	Average time LTI waits between receiving change sets from Log Reader.
LTI	Avg 16k LTM buffer utilization (%)	Average utilization of the 16K buffer used to store up LTL output to Replication Server (in percent).
LTI	Avg LTL commands/buffer	Average number of LTL commands per 16K buffer sent.
LTI	Input queue size	Number of change sets in the LTI input queue.
LTI	Output queue size	Number of distributes in the LTI output queue.
LTI	Last QID sent	Hex value of most recent origin queue ID sent to Replication Server.
LTI	Last transaction id sent	Hex value of most recent transaction ID sent to Replication Server.
LTI	Avg time to create distributes	Average time it takes to convert each change set to LTL.

- The ra_statistics command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also ra_status

ra_status

Description Returns the current state of the Replication Agent instance and a brief description of the last action taken.

Syntax ra_status

Examples

```
ra_status
```

This command returns the status of the Replication Agent instance, as shown below:

```
State  Action
 ------ ---------------------------
 ADMIN  Waiting for operator command
(1 row affected)
```

Usage

- The ra_status command returns the current state and a brief description of the last action taken. If the first word in the action description is "Transitioning," the Replication Agent instance is in transition between

states. Some Replication Agent commands are not valid when the Replication Agent instance is in state transition.

- Replication Agent states are:
  - *Admin* – in this state, the Replication Agent instance is running, but no connections are up. You can change any configuration parameter when the Replication Agent instance is in the *Admin* state.
  - *Replicating* – in this state, the Replication Agent Log Reader component is scanning the transaction log for operations to replicate from the primary database and the Log Transfer Interface component is sending LTL to the primary Replication Server.

  For more information about Replication Agent states, see "Replication Agent states" on page 47 in this book.
- The ra_status command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also quiesce, ra_statistics, resume, shutdown, suspend

ra_version

Description Returns the version of the Replication Agent instance, the host operating system version, and the JRE version.

Syntax ra_version

Examples

```
ra_version
```

This command returns a row as follows:

```
Sybase Replication Agent for Unix & Windows
NT,2000/12.5.0.301/P/generic/JDK 1.2.2/main/301/VM:
Sun Microsystems Inc. 1.2.2/OPT/Fri Mar  1 13:28:46 MST
2002
```

Usage The ra_version command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also pdb_version, ra_status, ra_version_all

ra_version_all

Description — Returns name, type, and version of the Replication Agent instance, version of the primary database server and communication driver, and version of the primary Replication Server and communication driver.

Syntax — ra_version_all

Examples

```
ra_version_all
```

This command returns the following rows:

```
Columns:        Component       Version

[ 1]    Instance:       rau_rat - IBM DB2 UDB

[ 2]    RepAgent:       Sybase Replication Agent for Unix & Windows
NT,2000/12.5.0.301/P/generic/JDK 1.2.2/main/301/VM: Sun Microsystems Inc.
1.2.2/OPT/Fri Mar  1 13:28:46 MST 2002

[ 3]    Java Runtime Environment:       Sun Microsystems Inc. 1.2.2

[ 4]    Primary Data Server:    DB2/NT 06.01.0000

[ 5]    PDS JDBC Driver:        DB2CLI.DLL 07.01.0000

[ 6]   RepServer:     Replication Server/12.5/P/Sun_svr4/OS  5.8/1/OPT/Thu
Feb 21 02:21:54 PST 2002

[ 7]    Sybase JDBC Driver:     com.sybase.jdbc2.jdbc.SybDriver 5.5
```

Usage

- When the ra_version_all command is invoked, all information returned by the command is recorded in the Replication Agent system log file.
- The ra_version_all command always returns information about the Replication Agent version and instance.
- The ra_version_all command returns information about the primary database server and communications driver and the primary Replication Server and communications driver only if the connections to those servers are properly configured. If a connection is not configured, or is configured incorrectly, the ra_version_all command returns <unknown> for the server associated with that connection.
- The ra_version_all command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also — pdb_version, ra_status, ra_version

resume

Description — Starts replication for the current transaction log.

Syntax — resume

Usage

- When resume is invoked, the Replication Agent starts replication operations.
  - The Log Reader component begins scanning the transaction log for operations to be replicated. The Log Reader begins scanning at the point specified by the LTM Locator.
  - When suitable operations are found, the Log Reader passes them (as change set data) to the input queue of the Log Transfer Interface (LTI) component.
  - As operations are placed in the input queue of the LTI, it generates the LTL output that will be sent to the primary Replication Server.
- The resume command returns an error under the following conditions:
  - the transaction log does not exist
  - the database connection for the primary database is not defined correctly in the primary Replication Server
  - the Replication Agent communication configuration parameters are not set correctly
- If the resume command is successful, the Replication Agent instance goes to the *Replicating* state.
- The resume command is valid only when the Replication Agent instance is in the *Admin* state.

See also — quiesce, ra_status, shutdown, suspend

shutdown

Description — Shuts down the Replication Agent instance.

Syntax — shutdown [immediate]

Parameters

immediate
The optional keyword that shuts down the Replication Agent instance immediately.

Usage

- When shutdown is invoked without specifying an option, a normal shutdown is performed.

  In a normal shutdown, Replication Agent first quiesces and then the application terminates. (See the description of the quiesce command on page 120 for more information on quiescing the Replication Agent instance.)

- When shutdown is invoked with the immediate keyword, an immediate shutdown is performed.

  In an immediate shutdown, the Replication Agent instance stops all processing, without regard to transactions in process or in transit, drops all connections immediately, and then terminates the application.

- The shutdown command with the immediate keyword is valid at any time, when the Replication Agent instance is in any state, including transition between states.

- The shutdown command with no keyword (normal shutdown) is valid when the Replication Agent instance is in either *Admin* or *Replicating* state, but not in state transition.

See also quiesce, ra_status, resume, suspend

suspend

Description Stops replication and puts the Replication Agent instance in the *Admin* state.

Syntax suspend

Usage

- When the suspend command is invoked, it stops all current Replication Agent processing.
  - The Log Reader component stops scanning the transaction log immediately and the Log Transfer Interface (LTI) component stops sending LTL to the primary Replication Server immediately.
  - Any data in Replication Agent internal queues is flushed without further processing.
  - The Replication Agent instance immediately drops all its connections to the primary database and the primary Replication Server.
  - The Replication Agent instance goes from the *Replicating* state to the *Admin* state.

- If the Replication Agent instance is in the *Admin* state, the suspend command returns an error message.
- The suspend command is valid only when the Replication Agent instance is in the *Replicating* state.

Note The action of the quiesce command is similar to that of the suspend command, except that the quiesce command allows pending change data in the Replication Agent internal queues to be processed before putting the Replication Agent instance in the *Admin* state.

See also quiesce, ra_status, resume, shutdown

test_connection

Description Tests all Replication Agent connections or a specified connection.

Syntax test_connection [*conn_name*]

Parameters *conn_name*

The keyword for a connection to be tested. Valid keywords are:

- RS – tests the connection to the primary Replication Server and its RSSD.
- PDS – tests the connection to the primary database server.

Examples **Example 1**

```
test_connection
```

This command tests all Replication Agent connections to both the primary Replication Server and the primary database.

Example 2

```
test_connection RS
```

This command tests Replication Agent connections to the primary Replication Server.

Usage

- When test_connection is invoked with no option, Replication Agent tests all connections.
- When test_connection is invoked with either the RS or PDS keyword, Replication Agent tests the specified connection.

- If the value of the use_rssd parameter is true, the test_connection command also tests the Replication Agent connection to the RSSD of the primary Replication Server when testing the Replication Server connection.
- Replication Agent tests a connection by attempting to log in to the corresponding server using the connection parameters recorded in the Replication Agent configuration file.

Note Even though test_connection returns a successful Replication Server connection, the Log Transfer Interface component still may not be able to successfully connect to the primary Replication Server because either the Replication Agent connection has not been created in the primary Replication Server, or the Replication Server user ID for the Replication Agent instance has not been granted connect source permission.

- The test_connection command returns the connection type and its status. If the connection status is failed, it indicates one of the following problems:
  - The connection parameters recorded in the Replication Agent configuration file are not correct, or
  - A network or communication error prevents the connection, or
  - The specified server is down.
- If the connection status is failed, you should check the system log file to determine the cause of the failure.
- The test_connection command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

See also ra_statistics

trace

Description Returns current trace flag settings or changes logging trace settings for Replication Agent.

Syntax trace [*flag*|all, *switch*]

Parameters *flag*
The name of the trace flag to change the setting for.

all
A keyword that allows you to apply a switch value to all trace flags.

switch

A boolean (true or false) value that enables or disables tracing for the trace point specified in the *flag* option.

Usage

- The trace command is intended for use by Sybase Technical Support engineers and when troubleshooting Replication Agent.
- When trace is invoked with no options specified, it returns the current settings for all the Replication Agent trace flags.
- When trace is invoked with the *flag* and *switch* options, it changes the setting of the trace point specified in the *flag* option and returns the current (new) setting for the specified trace point.
- When trace is invoked with the all keyword and a *switch* option, it sets all trace points to the value specified in the *switch* option and returns the current (new) setting for all trace points.
- Changes made with the trace command take effect immediately.
- When set to "true," tracing is enabled for the trace point identified by the specified trace flag. When set to "false," tracing is disabled for the trace point.
- Normal trace output is sent to the Replication Agent trace log file. You can use the log_trace_name command to find the name and path of the Replication Agent trace log file.
- Output from the *LTITRACELTL* trace point is sent to a separate LTL output log file (*LTITRACELTL.log*).
- Trace points identified as "system" flags cannot be changed with the trace command. Output from "system" flags is sent to the Replication Agent system log file and trace log file. You can use the log_system_name command to find the name and path of the Replication Agent system log file.
- Table 4-3 lists Replication Agent trace flags:

Table 4-3: Replication Agent trace flags

Trace Flag Name	Trace File	Description
BMGRTRACE	TRACE	When set to "true," this flag enables Bean Management event tracing.
ERROR	SYSTEM	A serious error has occurred, manual intervention may be required for recovery.

Trace Flag Name	Trace File	Description
FATAL	SYSTEM	A critical error has occurred, the application cannot function without manual intervention and will probably shut down.
INFORMATION	SYSTEM	Important information, no action required.
LATRC	TRACE	When set to “true,” this flag traces general Log Administrator operations.
LATRCSQL	TRACE	When set to “true,” this flag traces SQL conversations between Log Administrator and the primary database.
LRTRACE	TRACE	When set to “true,” this flag traces general execution of the Log Reader component.
LTITRACE	TRACE	When set to “true,” this trace flag enables tracing operations of the Log Transfer Interface component.
LTITRACELTL	TRACE	When set to “true,” this trace flag enables LTL statement tracing in the *LTITRACELTL.log* file.
LTMCI	TRACE	When set to “true,” causes tracing of LTM component interface invocations and LTM invocations of other components' interfaces.
LTMHL	TRACE	When set to “true,” causes highlights in the LTM execution path to be noted.
LTMSC	TRACE	When set to “true,” causes tracing of all Replication Agent state changes.
LTTRNC	TRACE	When set to “true,” causes tracing of transaction log truncation operations.
RACONTRC	TRACE	When set to “true,” causes tracing of connection and query execution.
RACONTRCSQL	TRACE	When set to “true,” causes tracing of SQL statements to be executed.
RATRACE	TRACE	When set to “true,” causes tracing of Replication Agent events.
STMTRACE	TRACE	When set to “true,” causes tracing of LTM state monitor events.
WARNING	SYSTEM	The system has suffered a minor problem; functionality is not affected, or the problem is recoverable.

- The trace command is valid when the Replication Agent instance is in either *Admin* or *Replicating* state.

Note Output from the LTITRACELTL trace point flag shows the LTL exactly as it is sent from Replication Agent to the primary Replication Server. If you want to view the LTITRACELTL output, make sure your viewer is capable of handling very long lines.

See also log_system_name, log_trace_name

CHAPTER 5

Sybase Replication Agent Configuration Parameters

This chapter describes the configuration parameters for Sybase Replication Agent and the Replication Agent configuration file.

This chapter includes the following sections:

- Configuration parameter overview
- Replication Agent configuration parameters
- Changing Replication Agent configuration parameters
- Replication Agent configuration file

Configuration parameter overview

Configuration parameters are user-configurable settings that control the behavior of a Replication Agent instance. Each Replication Agent instance has its own configuration file, which records the configuration information that instance needs to operate properly.

When a Replication Agent instance first starts up, it reads its configuration file. Thereafter, as long as that Replication Agent instance is running, the only time it accesses the configuration file is when the ra_config or ra_set_login command is invoked to change a configuration parameter.

When a configuration parameter is changed, Replication Agent saves that change in the configuration file, overwriting the entire file.

Note Because Replication Agent overwrites the entire configuration file every time the ra_config or ra_set_login command is invoked, Sybase recommends that you do not edit the configuration file.

Copying a Replication Agent configuration

When you create a new Replication Agent instance, using either the ra_admin utility or the Administrator (GUI) utility, you can specify that the new Replication Agent instance use configuration parameters that are already set for an existing Replication Agent instance.

Note When you choose to copy an existing configuration for a new Replication Agent instance, some of the configuration parameters in the existing configuration file are not copied to the new configuration file.

If you do not specify that an existing configuration be used when creating a new Replication Agent instance, the utility creates a new default configuration file, with default values for the configuration parameters, for the new Replication Agent instance.

For more information on using the ra_admin utility or the Administrator (GUI) utility, see Chapter 3, “Administering Sybase Replication Agent”.

Replication Agent configuration parameters

Table 5-1 lists all of the Replication Agent configuration parameters and gives a brief description of each parameter.

Table 5-1: Replication Agent configuration parameters

Parameter Name	Description
admin_port	Port number for Replication Agent administrative port.
column_compression	Use minimal column information.
compress_ltl_syntax	Use abbreviated LTL syntax.
connect_to_rs	Enable/disable connection from LTI to Replication Server.
dump_batch_timeout	Time to send incomplete buffer to Replication Server.
filter_maint_userid	Log Reader filters operations with maintenance user ID.
function_password	Password for user ID passed in LTL with replicated stored procedure invocations.
function_username	User ID passed in LTL with replicated stored procedure invocations.

Parameter Name	Description
log_directory	Directory where Replication Agent system log file is located.
log_trace_verbose	Switch on/off verbose mode in trace log file.
log_wrap	Number of 1K byte blocks written to log file before wrapping.
lr_update_trunc_point	Number of maintenance user operations skipped before Log Reader requests new LTM Locator.
lti_batch_mode	Switches on/off LTI batch mode.
lti_max_buffer_size	Maximum number of change sets stored in the LTI input buffer.
lti_update_trunc_point	Number of LTL commands sent before LTI requests new LTM Locator.
ltl_character_case	Case of database object names sent to Replication Server.
ltl_origin_time_required	Specifies whether to send origin_time command tag in LTL.
ltm_admin_pw	Password for Replication Agent administrative port.
ltm_admin_user	User ID for Replication Agent administrative port.
max_ops_per_scan	Maximum number of operations Log Reader will read in a single log scan.
pdb_auto_run_scripts	Automatic execution of SQL scripts used to create/remove transaction log objects and mark/unmark primary database objects.
pdb_convert_datetime	Converts native date/time formats to Sybase datetime format.
pdb_dflt_column_repl	Enables replication for LOB columns by default when table is marked.
pdb_dflt_object_repl	Enables replication by default when object is marked.
pdb_exception_handling	Allows primary database to continue operation if Replication Agent triggers fail.
pdb_xlog_device	Name of the primary database device.
pdb_xlog_prefix	Character string prefix used to identify transaction log objects.
pdb_xlog_prefix_chars	Non-alphabetic characters allowed in pdb_xlog_prefix.

Parameter Name	Description
pds_charset	Character set used on connection to primary database.
pds_connection_type	Type of connection to primary database server.
pds_database_name	Name of database replicated from primary database server.
pds_datasource_name	Data source name of database replicated from primary database server.
pds_host_name	Name of primary database server host machine.
pds_password	Password for user ID Replication Agent uses to access primary database server.
pds_port_number	Port number for primary database server.
pds_retry_count	Number of times to retry connection to primary database server.
pds_retry_timeout	Number of seconds to wait between connection retry attempts.
pds_server_name	Server name of primary database server.
pds_service_name	Name of database gateway service for primary database server.
pds_username	User ID that Replication Agent uses to access primary database server.
ra_retry_count	Number of times LTM attempts to get back to *Replicating* state after a failure.
ra_retry_timeout	Number of seconds to wait between LTM attempts to get back to *Replicating* state.
rs_host_name	Name of primary Replication Server host machine.
rs_packet_size	Network I/O packet size sent to Replication Server.
rs_password	Password for user ID Replication Agent uses to access Replication Server.
rs_port_number	Port number for primary Replication Server.
rs_retry_count	Number of times to retry connection to primary Replication Server.
rs_retry_timeout	Number of seconds to wait between connection retry attempts.
rs_source_db	Name of primary database identified to Replication Server.

Parameter Name	Description
rs_source_ds	Name of primary database server identified to Replication Server.
rs_username	User ID that Replication Agent uses to access primary Replication Server.
rssd_charset	Character set used in RSSD.
rssd_database_name	Name of RSSD database.
rssd_host_name	Name of RSSD host machine.
rssd_password	Password for user ID Replication Agent uses to access RSSD.
rssd_port_number	Port number for RSSD.
rssd_username	User ID that Replication Agent uses to access RSSD.
scan_sleep_increment	Number of seconds to increase Log Reader wait before next scan after finding no operations to replicate.
scan_sleep_max	Maximum number of seconds for Log Reader to wait before next scan after finding no operations to replicate.
skip_ltl_errors	LTI ignores error messages returned by Replication Server.
structured_tokens	LTI uses structured tokens when generating LTL output.
truncation_interval	Number of minutes to wait between automatic log truncations.
truncation_type	Method(s) of log truncation allowed.
use_rssd	Switches on/off access to RSSD for replication definitions.

admin_port

The client socket port number for the Replication Agent administration port.

Default

10000

Value

Any valid socket port number. (An integer from 1 to 65535.)

Comments

- When you create a Replication Agent instance, you must specify a client socket port number for its administration port. Client applications must use this port number to connect to the Replication Agent administration port.

- You must specify a client socket port number that does not conflict with any existing port numbers in use on the Replication Agent host machine.
- If you change the value of the admin_port parameter with the ra_config command, the new value is recorded in the configuration file immediately, but you must shut down and restart the Replication Agent instance to make the new port number take effect.
- After you change the value of the admin_port parameter with the ra_config command, the next time you log in to the Replication Agent administration port, you must use the new socket port number.

column_compression

Determines whether the Log Transfer Interface component sends all columns in after images or only columns that were changed in an update operation in the after images.

Default: true

Values: true – The Log Transfer Interface sends only minimal column information (only changed columns in after images) in the generated LTL for update operations.

false – The Log Transfer Interface sends complete column information (all columns in after images) in the generated LTL for update operations.

Comments:

- When the column_compression parameter is set to false, the Log Transfer Interface sends complete before and after images in the generated LTL, even for columns in which no data changed as a result of an update operation.
- When the column_compression parameter is set to true, the Log Transfer Interface sends LTL with only the columns that changed as a result of an update operation in after images. Columns in which no data changed as a result of the update operation are not sent in the after images.
- In general, setting the value of the column_compression parameter to true provides better Replication Agent throughput.

compress_ltl_syntax

Determines whether the Log Transfer Interface component compresses LTL statements using abbreviated syntax.

Default: true

Values: true – The Log Transfer Interface compresses LTL statements using abbreviated syntax.

false – The Log Transfer Interface uses uncompressed LTL syntax when generating LTL statements.

Comments:

- In general, setting the value of the compress_ltl_syntax parameter to true provides better Replication Agent throughput.
- See the *Replication Server Administration Guide* for more information about the abbreviated LTL syntax.

connect_to_rs

Enables or disables the Replication Agent LTI connection to the primary Replication Server.

Default: true

Values: true – The Replication Agent connection to the primary Replication Server is enabled.

false – The Replication Agent connection to the primary Replication Server is disabled.

Comments:

- When the value of the connect_to_rs parameter is false, no replication will occur. When the Replication Agent instance enters the *Replicating* state:
  - a "dummy" connection emulates a real connection to the primary Replication Server
  - the value of the LTM Locator stored in the Replication Agent transaction log is set to all zeros
  - the value of the maintenance user ID is FRED
- You can use the connect_to_rs parameter to temporarily disable the connection to the primary Replication Server for testing.
- When the value of the connect_to_rs parameter is false, you can put the Replication Agent instance in the *Replicating* state, set the value of the LTITRACELTL trace flag to true, and view the LTL output that would have been sent to the primary Replication Server if the value of the connect_to_rs parameter were true.

- During normal Replication Agent operation, the value of the connect_to_rs parameter is true.

dump_batch_timeout

Specifies the time interval to wait before sending the contents of the Log Transfer Interface buffer to the primary Replication Server, even though the buffer is not full.

Default: 5

Value: An integer from 1 to 60.

Comments:

- The value of the dump_batch_timeout parameter is the number of seconds from the time the previous Log Transfer Interface buffer was sent to the primary Replication Server until the next buffer will be sent.
- The dump_batch_timeout parameter has no effect if the value of the lti_batch_mode parameter is false.

filter_maint_userid

Determines whether the Replication Agent Log Reader component filters database operations applied in the primary database by the Replication Server maintenance user ID.

Default: true

Values: true – The Log Reader component filters out database operations applied by the Replication Server maintenance user ID.

false – The Log Reader component replicates database operations regardless of the user ID that applied the operation.

Comments:

- The filter_maint_userid configuration parameter is provided to support bidirectional replication, wherein the primary database is also a replicate database that has transactions applied to it by a Replication Server.
- If the value of the filter_maint_userid parameter is true, database operations applied by the Replication Server maintenance user ID are *not* replicated. The Log Reader component filters out the Replication Server maintenance user ID when it reads the transaction log.

- If the value of the filter_maint_userid parameter is false, database operations applied by the Replication Server maintenance user ID are replicated.
- The Replication Server maintenance user ID is specified when the database connection for the primary database is created in the primary Replication Server.

function_password

The password included in LTL for replicated stored procedures.

Default: "" (empty string)

Values: A valid password.

Comments:

- The value of the function_password parameter can be up to 30 characters.
- The value of the function_password parameter is the password for the user ID specified in the function_username parameter.
- The value of the function_password parameter is encrypted in the Replication Agent configuration file.

function_username

The user ID included in LTL for replicated stored procedures.

Default: sa

Values: A valid user ID.

Comments:

- The value of the function_username parameter is sent in the LTL for all replicated stored procedures in the primary database.
- The value of the function_password parameter is the password for the user ID specified in the function_username parameter.

log_directory

The directory for system log files on the Replication Agent host.

Default: Current directory on the Replication Agent host machine from where the Replication Agent instance was started.

Value — Any valid directory path on the Replication Agent host machine.

Comments

- When a Replication Agent instance is created, a log directory is created as part of the instance directory structure and the value of the log_directory parameter points to that directory, for example:

  ```
  C:\%SYBASE%\rax-12_5\inst_name\log\
  ```

  where *%SYBASE%* is the name of the Replication Agent installation directory and *inst_name* is the name of the Replication Agent instance.
- If you specify any valid directory path as the value of the log_directory parameter, Replication Agent places its system and error log files in the directory you specify.
- If you specify the "default" value of the log_directory parameter by using the default keyword in the ra_config command, Replication Agent places its system and error log files in the current directory from which the Replication Agent instance was started.

log_trace_verbose

Determines whether Replication Agent provides additional detailed diagnostic information in trace log files.

Default — false

Values — true – Replication Agent displays detailed diagnostic information in trace log files.

false – Replication Agent displays concise diagnostic information in trace log files.

Comment — Detailed diagnostic information is intended for troubleshooting only with assistance from Sybase Technical Support.

log_wrap

The amount of log output written in 1k byte blocks before wrapping log files.

Default — 500

Value — An integer greater than or equal to 500.

Comments

- The value of the log_wrap parameter is the number of 1k byte blocks of log output that will be written by Replication Agent before it wraps the system log file or the trace log file.
- Larger values for the log_wrap parameter will result in more history retained in the Replication Agent system log file and trace log file. Smaller values for the log_wrap parameter will result in smaller log files.
- When the log wraps, Replication Agent copies the old log file to a file with the same name and with the *.old* extension. For example, if the log file is named *ra_ny.log*, the file created when the log wraps is named *ra_ny.old*.

lr_update_trunc_point

The number of maintenance operations or unmarked table references that can be skipped before requesting a new truncation point from the primary Replication Server.

Default: 1000

Value: An integer in the range of 1 to 10000.

Comments

- The lr_update_trunc_point configuration parameter is provided to allow logged operations that are not replicated (maintenance user operations and references to unmarked tables) to be truncated from the transaction log.
- The value of the lr_update_trunc_point parameter is the number of maintenance operations or unmarked table references that the Log Reader component can process before the Replication Agent instance requests a new truncation point (LTM Locator) from the primary Replication Server.
- If the value of the truncation_type parameter is locator_update, setting the value of the lr_update_trunc_point parameter to a lower number will cause automatic log truncation to occur more frequently.
- If the value of the lr_update_trunc_point parameter is 0 (zero), Replication Agent does not request a new truncation point when processing maintenance operations or unmarked table references.
- When Replication Agent processes transactions for replication, it automatically requests a new truncation point from the primary Replication Server based on the value of the lti_update_trunc_point parameter.

  When Replication Agent processes maintenance operations or unmarked table references, and the value of the filter_maint_userid parameter is true,

Replication Agent does not send LTL to the primary Replication Server for those operations, and the automatic request for a new truncation point based on the value of the lti_update_trunc_point parameter never takes place.

The lr_update_trunc_point parameter allows Replication Agent to periodically request a new truncation point from the primary Replication Server, even when no LTL output is sent. This provides faster recovery by ensuring that Replication Agent has a current value for the LTM Locator.

lti_batch_mode

Determines whether the Log Transfer Interface component processes LTL output in batch mode.

Default: true

Values: true – The Log Transfer Interface component fits as many LTL commands into a 16k byte buffer as it can before it sends output to the primary Replication Server.

false – The Log Transfer Interface component sends individual LTL commands for each change set in its input queue.

Comments:

- If the value of the lti_batch_mode parameter is true, the Log Transfer Interface component sends the buffer contents to the primary Replication Server when the time interval specified in the dump_batch_timeout parameter expires, even if the buffer is not full.
- If the value of the lti_batch_mode parameter is true and an individual distribute command exceeds the size of the 16k byte buffer, the primary Replication Server returns an error and drops the database connection to the Replication Agent instance.
- In general, setting the value of the lti_batch_mode parameter to true provides better Replication Agent throughput.

lti_max_buffer_size

Determines the maximum number of change sets that can be stored in the incoming buffer of the Log Transfer Interface component.

Default: 5000

Value — An integer in the range of 1000 to 100000.

Comments

- The value of the lti_max_buffer_size parameter is the maximum number of change sets processed by the Log Reader component that can be stored in the incoming buffer of the Log Transfer Interface (LTI) component.
- The LTI incoming buffer is a bounded buffer that blocks Log Reader processing when it gets full.

lti_update_trunc_point

The number of LTL commands sent before requesting a new truncation point from Replication Server.

Default — 1000

Value — An integer from 1 to 100000.

Comments

- The value of the lti_update_trunc_point parameter is the number of LTL commands that Replication Agent sends to the primary Replication Server before Replication Agent requests a new truncation point (LTM Locator) from the primary Replication Server.
- Lower numbers will cause Replication Agent to request a truncation point from Replication Server more often.
- If the value of the truncation_type parameter is locator_update, setting the value of the lti_update_trunc_point parameter to a lower number will cause automatic log truncation to occur more frequently.
- If the value of the lti_update_trunc_point parameter is too low, it can slow down the operation of Replication Agent.
- If Replication Agent operates in an unreliable network environment, setting the value of the lti_update_trunc_point parameter to a lower number can result in quicker recovery.

ltl_character_case

Specifies the case used for database object names in the LTL sent to the primary Replication Server.

Default — asis

Values

asis – Passes database object names in the LTL output in the same case as they are returned from the primary database, or (if the value of the use_rssd parameter is true) in the case they are specified in replication definitions.

lower – Passes all object names in the LTL output in lower case, regardless of how they are returned from the primary database or specified in replication definitions.

upper – Passes all object names in the LTL output in upper case, regardless of how they are returned from the primary database or specified in replication definitions.

Comments

- The ltl_character_case configuration parameter allows you to customize the handling of database object names in the LTL output to work with existing replication definitions that specify the object names differently than the way the primary database returns them.
- If the value of the use_rssd parameter is false and the value of the ltl_character_case parameter is asis, database object names are passed in the LTL output in the same case as they are returned from the primary database.
- If the value of the use_rssd parameter is true and the value of the ltl_character_case parameter is asis, database object names are passed in the LTL output in the same case as they are specified in the replication definitions (stored in the RSSD of the primary Replication Server).
- If object names are specified as all lower case in replication definitions, set the value of the ltl_character_case parameter to lower.
- If object names are specified as all upper case in replication definitions, set the value of the ltl_character_case parameter to upper.
- If you want to pass database object names with mixed case (for example, TableName), then you must set the value of the ltl_character_case parameter to asis.

ltl_origin_time_required

Determines whether the Log Transfer Interface component specifies the time that it generates LTL commands in the origin_time command tag.

Default

false

Values

true – The Log Transfer Interface component includes the origin_time command tag in the generated LTL.

false – The Log Transfer Interface component does not include the origin_time command tag in the generated LTL.

Comments

- If a Replication Server function string checks for the origin_time command tag, you should set the value of the ltl_origin_time_required parameter to true.
- The datetime value placed in the Replication Server origin_time command tag is the time when the original replicated operation was recorded in the transaction log, not the time it was scanned by the Replication Agent Log Reader component.
- Setting the value of the ltl_origin_time_required parameter to false provides better Replication Agent throughput.
- If you use Replication Server Manager to report latency, you must set the value of the ltl_origin_time_required parameter to true.

ltm_admin_pw

The Replication Agent administrative user password used to gain access to the Replication Agent administration port.

Default

"" (empty string)

Value

A valid password.

Comments

- The value of the ltm_admin_pw parameter is the password for the user ID authorized to log in to the Replication Agent administration port.
- The value of the ltm_admin_pw parameter is encrypted in the Replication Agent configuration file.
- To change the value of the ltm_admin_pw parameter, you must use the ra_set_login command.
- When you change the value of the ltm_admin_pw parameter with the ra_set_login command, the new value is recorded immediately and you must use the new password next time you log in to the Replication Agent administration port.

ltm_admin_user

The Replication Agent administrative user ID.

Default: sa

Value: A valid user ID.

Comments:

- The value of the ltm_admin_user parameter is the user ID authorized to log in to the Replication Agent administration port.
- To change the value of the ltm_admin_user parameter, use the ra_set_login command.
- If you change the value of the ltm_admin_user parameter with the ra_set_login command, the new value is recorded immediately, but you must shut down and restart the Replication Agent instance to make the new administrative user ID take effect.
- After you change the value of the ltm_admin_user parameter with the ra_set_login command, the next time you log in to the Replication Agent administration port, you must use the new administrative user ID.
- Exactly one administrative user ID is valid at any time.

max_ops_per_scan

The number of primary database operations requested from the transaction log by the Log Reader component during each log scan operation.

Default: 1000

Values: An integer from 25 to 2147483647.

Comments:

- The value of the max_ops_per_scan parameter is the maximum number of database operations that will be read from a trigger-based Replication Agent transaction log during each Log Reader scan operation.
- The Log Reader component will always read at least one transaction in each scan of the transaction log, regardless of how many operations are in the transaction.

  For example, if the value of the max_ops_per_scan parameter is 1000 and a transaction contains 1200 operations, the Log Reader will actually read all 1200 operations in one scan when it reads that transaction.
- For information about how the max_ops_per_scan parameter affects the log-based Replication Agent for UDB, see “Read buffer size” on page 206.

pdb_auto_run_scripts

Determines whether Replication Agent automatically runs transaction log creation and removal scripts and object marking and unmarking scripts at the primary database.

Default: true

Values: true – Replication Agent automatically runs transaction log scripts at the primary database.

false – Replication Agent saves the scripts, but does not automatically run transaction log scripts at the primary database.

Comments:

- When the pdb_xlog command is invoked to create or remove the transaction log, Replication Agent generates a script to create or remove the transaction log base objects.
- When either the pdb_setrepproc or pdb_setreptable command is invoked to mark or unmark an object in the primary database, Replication Agent generates a script to create or remove the transaction log objects necessary for object marking.
- The transaction log scripts are always saved in a file. Log creation and removal scripts are saved in files named *create.sql* and *remove.sql*. Object marking and unmarking scripts are saved in files named *mark.sql* and *unmark.sql*.
- To mark or unmark a primary database object when the pdb_auto_run_scripts parameter is set to false, you must run each script manually after Replication Agent creates it and saves the file.

pdb_convert_datetime

Determines whether Replication Agent converts the primary database native temporal datatypes to the Sybase datetime format.

Default: false

Values: true – Replication Agent converts all data in the primary database native date/time datatype(s) to the Sybase datetime format.

false – Replication Agent replicates data in the primary database native date/time datatype(s) as character strings using the char datatype.

Comments

- The pdb_convert_datetime parameter is provided for backward compatibility with previous versions of Replication Agents and Replication Server. If you use Replication Server version 12.0 or later, Sybase recommends that you use the Replication Server heterogeneous datatype support (HDS) for all datatype conversion and translation.
- Replication Agent checks the value of the pdb_convert_datetime parameter at the time an object is marked for replication. Transaction log objects that support replication of the marked object are constructed to provide the desired date format. If you change the value of the pdb_convert_datetime parameter after an object is marked, it has no effect on the marked object. To change the date/time datatype conversion for a marked object, you must unmark the object, change the value of the pdb_convert_datetime parameter, then re-mark the object.
- The actual conversion of the date/time datatype takes place at the time an operation is recorded in the transaction log.
- Any missing component in the primary database native date/time datatype format is treated as an implied 0 (zero) value when it is converted to the Sybase datetime format.
- When the value of the pdb_convert_datetime parameter is true, the replication definition for each table should specify that the declared datatype for all date/time columns is datetime.
- If the value of the pdb_convert_datetime parameter is false, the Replication Agent sends date/time columns to the primary Replication Server as default-sized character strings. The default date/time character string length varies for each database and each date/time datatype:
  - DB2 Universal Database: DATE = char(10), TIME = char(8), TIMESTAMP = char(16)
  - Informix: date = char(9), datetime = char(25)
  - Microsoft SQL Server: datetime or smalldatetime = char(23), timestamp = binary(8)
  - Oracle: DATE = char(11)

  If the primary database date/time datatype format requires a longer string to replicate correctly, you can either set the value of the pdb_convert_datetime parameter to true, or modify the trigger-based Replication Agent table marking script (mark.sql) to set up larger shadow table date/time columns.

- If all date/time values replicated from the primary database are replicated using the Sybase datetime datatype, then the value of the pdb_convert_datetime parameter should be set to true.
- Replication Agent date/time datatype conversion does not work with LOB column replication, unless either of the following conditions exist:
  - There are no date/time columns in the tables that have LOB column replication enabled, or
  - The primary keys in tables that have LOB column replication enabled do not contain date/time datatypes.

  Otherwise, if you use the pdb_setrepcol command to enable LOB column replication, you must set the value of the pdb_convert_datetime parameter to false.
- Sybase recommends that the value of the pdb_convert_datetime parameter be set to false for better Replication Agent logging performance and optimal datatype handling.

pdb_dflt_column_repl

Determines whether LOB column replication is enabled by default at the time a table is marked.

Default: false

Values: true – LOB column replication is enabled by default (automatically) at the time a table is marked.

false – LOB column replication is disabled by default at the time a table is marked.

Comments:

- If the pdb_dflt_column_repl parameter is set to false at the time a table is marked for replication, no transactions that affect LOB columns in the table can be replicated until replication is explicitly enabled with the pdb_setrepcol command.
- You can use the pdb_setrepcol command to enable or disable replication for all LOB columns in all marked tables at once.
- When replication is disabled for a LOB column, any part of an operation that affects that column will not be recorded in the transaction log.

pdb_dflt_object_repl

Determines whether replication is enabled by default at the time an object (table or stored procedure) is marked.

Default: true

Values: true – Replication is enabled by default (automatically) at the time an object is marked.

false – Replication is disabled by default at the time an object is marked.

Comments:

- If the pdb_dflt_object_repl parameter is set to false at the time a table is marked for replication, no transactions can be replicated from that table until replication is enabled with the pdb_setreptable command.
- If the pdb_dflt_object_repl parameter is set to false at the time a stored procedure is marked for replication, no invocations of that stored procedure can be replicated until replication is enabled with the pdb_setrepproc command.
- You can use the pdb_setrepproc or pdb_setreptable command to enable or disable replication for all marked stored procedures or tables at once.
- When replication is disabled for a table, no operations that affect that marked table will be recorded in the transaction log.
- When replication is disabled for a stored procedure, no invocations of that stored procedure will be recorded in the transaction log.

pdb_exception_handling

Determines how Replication Agent trigger errors are handled in the primary database.

Default: false

Value: true – If an error occurs during trigger execution, the error is logged in the exceptions table and the transaction in the primary database continues without being recorded in the Replication Agent transaction log.

false – If an error occurs during trigger execution, the transaction in the primary database fails, reporting the error encountered to the entity executing the transaction.

Comments:

- When the value of the pdb_exception_handling parameter is true, trigger errors are logged in the exceptions table and transactions against the marked object are allowed to continue without being captured in the

Replication Agent transaction log. Although replication stops for all marked objects, the operation of the primary database is not interrupted by the failure of a Replication Agent trigger.

Note If the value of the pdb_exception_handling parameter is true and replication is interrupted by a trigger error, transactions will continue to occur in the primary database, but they will not be replicated. When transactions in the primary database are not replicated, the replicate database must be rematerialized before replication can resume.

- Once replication is interrupted by the failure of a Replication Agent trigger, it cannot be resumed until the cause of the trigger error is corrected.
- When the value of the pdb_exception_handling parameter is false, a trigger error will cause the transaction in the primary database to fail. The primary database reports the transaction failure to the entity that executed the transaction. Although operation of the primary database is interrupted for all marked objects, the integrity of the replication system is not affected (i.e., no transactions can occur without being replicated).
- If maintaining the availability of the primary database is a higher priority than maintaining integrity of the replicate database, set the value of the pdb_exception_handling parameter to true.
- If maintaining the integrity of the replicate database is a higher priority than maintaining the availability of the primary database, set the value of the pdb_exception_handling parameter to false.

pdb_xlog_device

The primary database device where transaction log objects should be created.

Default: NULL

Value: A valid primary database device name or NULL.

Comments:

- The value of the pdb_xlog_device parameter is the device specification of the primary database device to be used in SQL scripts generated by the Replication Agent to create transaction log objects.
- The pdb_xlog_device parameter allows you to specify a single device on which all Replication Agent transaction log objects will be created.

- If the value of the pdb_xlog_device parameter is NULL, no device is specified in the SQL create statements. If no device is specified in the SQL create statements, transaction log objects are placed on the primary database server's system-defined default device.

pdb_xlog_prefix

The prefix used in database object names to identify Replication Agent transaction log objects.

Default

ra_

Value

A character string of 1 to 3 characters.

Comments

- When Replication Agent generates database object names for any transaction log component in the primary database, it uses the value of the pdb_xlog_prefix parameter as an object name prefix.
- Replication Agent uses the value of the pdb_xlog_prefix parameter to recognize its transaction log objects in the primary database. Therefore, if you change the value of the pdb_xlog_prefix parameter after you create transaction log objects, Replication Agent will not be able to find the transaction log objects created with the previous prefix string.
- The value of the pdb_xlog_prefix_chars parameter specifies the non-alphabetic characters that can be used in the prefix string.

pdb_xlog_prefix_chars

The non-alphabetic characters allowed for the prefix used in database object names to identify Replication Agent transaction log objects.

Default

_#@ (DB2 Universal Database)

_ (Informix)

_$#@ (Microsoft SQL Server)

_#$ (Oracle)

Value

A string of characters with no separators.

Comments

- The default value of the pdb_xlog_prefix_chars parameter depends on the type of primary database the Replication Agent instance was created for.

The default value is based on the standard non-alphabetic characters allowed by each database.

- When you set or change the value of the pdb_xlog_prefix_chars parameter, the new value replaces any existing value; it does not add or append the new value to a previous value.
- When the value of the pdb_xlog_prefix parameter is set using the ra_config command, all non-alphabetic characters specified on the command line are validated against the characters in the value of the pdb_xlog_prefix_chars parameter.
- Alphabetic characters a-z are always valid in the value of the pdb_xlog_prefix parameter and need not be specified.
- Sybase Replication Agent does not support delimited names for transaction log objects, so you cannot use a space character in the value of the pdb_xlog_prefix parameter.
- The value you specify for the pdb_xlog_prefix_chars parameter is not validated. There are no restrictions on the characters you can include in the value of the pdb_xlog_prefix_chars parameter.

Note The primary database server may restrict the characters used in certain positions in a database object name. Refer to the documentation for your primary database server for more information.

pds_charset

The character set used for communication over the primary data server connection.

Default: "" (empty string)

Value: A valid character set name.

Comments:

- The value of the pds_charset parameter is the name of the character set used when Replication Agent communicates with the primary database server.
- The pds_charset parameter should be used only if the primary database JDBC driver requires it.

pds_connection_type

The type of database connectivity driver used to connect Replication Agent to the primary database.

Default

<not_configured> (One of the following values is set automatically when the Replication Agent instance is created.)

Values

IFXJDBC – Replication Agent uses the Informix JDBC driver to connect to the primary Informix database.

MSMERJDBC – Replication Agent uses the DataDirect JDBC 2.0 driver for Microsoft SQL Server to connect to the primary Microsoft SQL Server database.

ORAJDBC – Replication Agent uses the Oracle JDBC driver to connect to the primary Oracle database.

UDBODBC – Replication Agent uses the DB2 Universal Database JDBC driver to connect to the primary database in DB2 Universal Database.

Comments

- The value of the pds_connection_type parameter is set automatically at the time a Replication Agent instance is created. The specific value depends on the type of Replication Agent instance created.
- The value of the pds_connection_type parameter determines which of several other Replication Agent configuration parameters related to the primary database connection must also have values specified.
- If the value of the pds_connection_type parameter is IFXJDBC, then corresponding values must be specified for the pds_host_name, pds_port_number, pds_database_name, and pds_server_name parameters.
- If the value of the pds_connection_type parameter is MSMERJDBC, then corresponding values must be specified for the pds_host_name, pds_port_number, pds_database_name, and pds_server_name parameters.
- If the value of the pds_connection_type parameter is ORAJDBC, then corresponding values must be specified for the pds_host_name, pds_port_number, and pds_database_name parameters.
- If the value of the pds_connection_type parameter is UDBODBC, then corresponding values must be specified for the pds_database_name and pds_datasource_name parameters.

pds_database_name

The name of the primary database.

Default: <not_configured>

Value: A valid database name.

Comments:

- The value of the pds_database_name parameter is the name of the primary database on the primary database server.
- Some primary database servers may not support multiple databases in a single instance of the database server. In that case, the value of the pds_database_name parameter should be the instance name of the database server instance.

pds_datasource_name

The data source name (DSN) specified for the ODBC driver used to communicate with the primary database.

Default: <not_configured>

Value: A valid ODBC data source name.

Comments:

- The value of the pds_datasource_name parameter is the data source name (DSN) of the ODBC driver for the primary database.
- If the value of the pds_connection_type parameter is UDBODBC, the value of the pds_datasource_name parameter must be the database alias of the primary database in the DB2 Universal Database server.
- This parameter is used only if the value of the pds_connection_type parameter is UDBODBC.

pds_host_name

The name of the host machine on which the primary database server resides.

Default: <not_configured>

Value: A valid host name.

Comment: The value of the pds_host_name parameter is the name of the host machine on which the primary database server resides.

pds_password

The password for the user ID that Replication Agent uses to access the primary database.

Default: "" (empty string)

Value: A valid password.

Comments:

- The value of the pds_password parameter is the password for the user ID that Replication Agent uses to access the primary database.
- The value of the pds_password parameter is encrypted in the Replication Agent configuration file.

pds_port_number

The client socket port number for the primary database server.

Default: 1111

Value: A valid socket port number.

Comment: The value of the pds_port_number parameter is the client socket port number for the primary database server.

pds_retry_count

The number of times to retry establishing a connection to the primary database.

Default: 5

Value: An integer from 0 to 2147483647.

Comments:

- The value of the pds_retry_count parameter is the number of times that Replication Agent will try to establish a connection to the primary database after a connection failure.
- Sybase recommends a setting of 5 for this parameter.

pds_retry_timeout

The number of seconds to wait between retry attempts to connect to the primary database.

Default	10
Value	An integer from 0 to 3600.
Comment	The value of the pds_retry_timeout parameter is the number of seconds that Replication Agent will wait between its retry attempts to establish a connection to the primary database after a connection failure.

pds_server_name

The server name of the primary database server.

Default	<not_configured>
Value	A valid server name.
Comment	The value of the pds_server_name parameter is the server name of the primary database server.

pds_service_name

The service name of the primary database gateway server.

Default	<not_configured>
Value	A valid service name.
Comments	• The value of the pds_service_name parameter is the service name of the primary database gateway server. • This parameter is not used in Sybase Replication Agent version 12.0 or later.

pds_username

The user ID that Replication Agent uses to access the primary database.

Default	<not_configured>
Value	A valid user ID.
Comments	• The value of the pds_username parameter is the user ID that Replication Agent uses to access the primary database.

- The user ID that Replication Agent uses to access the primary database must be defined in the primary database with appropriate privileges or authority in the primary database.
- Replication Agent uses this user ID to access primary database objects and to create, remove, and manage its transaction log objects in the primary database.

ra_retry_count

The number of times LTM attempts to re-establish replication after a failure.

Default: 2

Value: An integer greater than 0.

Comments:

- The value of the ra_retry_count parameter is the number of times that the LTM component will try to get the Replication Agent instance back into *Replicating* state after a failure causes the Replication Agent instance to go from the *Replicating* state to the *Admin* state.
- When a connection failure occurs, Replication Agent attempts to re-establish the lost connection. Replication Agent uses the settings of the pds_retry_count and pds_retry_timeout parameters if the connection to the primary database is lost. Replication Agent uses the settings of the rs_retry_count and rs_retry_timeout parameters if the connection to the primary Replication Server is lost.

  If Replication Agent is unable to re-establish a lost connection after the number of retries specified in either the pds_retry_count or rs_retry_count parameters, then the Replication Agent instance goes to the *Admin* state and the LTM component attempts to return the Replication Agent instance to the *Replicating* state, according to the settings of the ra_retry_count and ra_retry_timeout parameters.

ra_retry_timeout

The number of seconds to wait between LTM attempts to re-establish replication after a failure.

Default: 10

Value: An integer greater than 0.

Comment — The value of the ra_retry_timeout parameter is the number of seconds that the LTM component will wait between its attempts to get the Replication Agent instance back into *Replicating* state after a failure causes the Replication Agent instance to go from the *Replicating* state to the *Admin* state.

rs_host_name

The name of the primary Replication Server host machine.

Default — <not_configured>

Value — A valid host name.

Comment — The value of the rs_host_name parameter is the name of the host machine for the primary Replication Server (the Replication Server to which Replication Agent sends LTL output).

rs_packet_size

The maximum size (in bytes) of the network packets sent to the primary Replication Server.

Default — 2048

Value — An integer from 512 to 8192.

Comments

- The value of the rs_packet_size parameter is the maximum size (in bytes) of the network packets handled by the TCP/IP network protocol.
- The rs_packet_size parameter is equivalent to the Replication Server rsi_packet_size parameter.
- When the network packet size is smaller, more packets must be processed to transmit a given amount of data from the Replication Agent to the primary Replication Server. When the network packet size is larger, more system resources are consumed to process the packets.
- The optimum value of the rs_packet_size parameter is based on the nature of the typical data replicated. If the typical transaction operation is very large, a larger maximum packet size is more efficient.
- A larger value of the rs_packet_size parameter is also more efficient if the value of the lti_batch_mode parameter is true.

rs_password

The password for the user ID that Replication Agent uses to log in to the primary Replication Server.

Default: "" (empty string)

Value: A valid password.

Comments:

- The value of the rs_password parameter is the password for the user ID that Replication Agent uses to log in to the primary Replication Server (the Replication Server to which Replication Agent sends LTL output).
- The value of the rs_password parameter is encrypted in the Replication Agent configuration file.

rs_port_number

The client socket port number of the primary Replication Server.

Default: 1111

Value: A valid socket port number.

Comment: The value of the rs_port_number parameter is the client socket port number of the primary Replication Server (the Replication Server to which Replication Agent sends LTL output).

rs_retry_count

The number of times to retry establishing a connection to the primary Replication Server.

Default: 5

Value: An integer greater than 0.

Comments:

- The value of the rs_retry_count parameter is the number of times that Replication Agent will try to establish a connection to the primary Replication Server after a connection failure.
- Sybase recommends a setting of 5 for this parameter.

rs_retry_timeout

The number of seconds to wait between retry attempts to connect to the primary Replication Server.

Default: 10

Value: An integer greater than 0.

Comment: The value of the rs_retry_timeout parameter is the number of seconds that Replication Agent will wait between its retry attempts to establish a connection to the primary Replication Server after a connection failure.

rs_source_db

The name of the primary database that Replication Agent uses to tell the primary Replication Server which log it is sending.

Default: <not_configured>

Value: A valid database name.

Comments:

- The value of the rs_source_db parameter is the name of the primary database by which the primary Replication Server recognizes the primary database transaction log.
- The value of the rs_source_db parameter must match the name of the database specified in the Replication Server create connection command used to create the Replication Agent connection in the primary Replication Server.

rs_source_ds

The name of the data server that Replication Agent uses to tell the primary Replication Server which log it is sending.

Default: <not_configured>

Value: A valid server name.

Comments:

- The value of the rs_source_ds parameter is the name of the data server (primary database server) by which the primary Replication Server recognizes the primary database transaction log.
- The value of the rs_source_ds parameter must match the name of the data server specified in the Replication Server create connection command used

to create the Replication Agent connection in the primary Replication Server.

- The value of the rs_source_ds parameter should *not* be the same as the name of the Replication Agent instance.

rs_username

The user ID that Replication Agent uses to log in to the primary Replication Server.

Default: <not_configured>

Value: A valid user ID.

Comments:

- The value of the rs_username parameter is the user ID that Replication Agent uses to log in to the primary Replication Server (the Replication Server to which Replication Agent sends LTL output).
- The value of the rs_password parameter is the password for the user ID specified by the rs_username parameter.
- The user ID that Replication Agent uses to log in to the primary Replication Server must have connect source permission in the primary Replication Server.

rssd_charset

The character set used for communication over the connection to the RSSD of the primary Replication Server.

Default: "" (empty string)

Value: A valid character set name.

Comments:

- The value of the rssd_charset parameter is the name of the character set used when Replication Agent communicates with the RSSD of the primary Replication Server.
- You can use any character set supported by the Java VM on the Replication Agent host machine.

rssd_database_name

The database name of the RSSD of the primary Replication Server.

Default: <not_configured>

Value: A valid database name.

Comments:

- The value of the rssd_database_name parameter is the database name of the RSSD of the primary Replication Server.
- The rssd_database_name parameter need not be set if the use_rssd parameter is set to false.

rssd_host_name

The name of the machine on which the RSSD of the primary Replication Server resides.

Default: <not_configured>

Value: A valid host name.

Comments:

- The value of the rssd_host_name parameter is the host name of the machine on which the RSSD of the primary Replication Server resides.
- The rssd_host_name parameter need not be set if the use_rssd parameter is set to false.

rssd_password

The password for the user ID that Replication Agent uses to access the RSSD of the primary Replication Server.

Default: "" (empty string)

Value: A valid password.

Comments:

- The value of the rssd_password parameter is the password for the user ID that Replication Agent uses to access the RSSD of the primary Replication Server.
- The value of the rssd_password parameter is encrypted in the Replication Agent configuration file.
- The rssd_password parameter need not be set if the use_rssd parameter is set to false.

rssd_port_number

The client socket port number of the RSSD of the primary Replication Server.

Default: 1111

Value: A valid socket port number.

Comments:

- The value of the rssd_port_number parameter is the client socket port number of the RSSD of the primary Replication Server.
- The rssd_port_number parameter need not be set if the use_rssd parameter is set to false.

rssd_username

The user ID that Replication Agent uses to access the RSSD of the primary Replication Server.

Default: <not_configured>

Value: A valid user ID.

Comments:

- The value of the rssd_username parameter is the user ID that Replication Agent uses to access the RSSD of the primary Replication Server.
- The rssd_username parameter need not be set if the use_rssd parameter is set to false.

scan_sleep_increment

The number of seconds to add to each wait before checking the transaction log for transactions to be replicated after a previous scan yields no such transaction.

Default: 5

Value: An integer from 0 to 3600.

Comments:

- The value of the scan_sleep_increment parameter is the number of seconds added to each wait before the Log Reader component checks the transaction log in the primary database for a transaction to be replicated after a previous scan yields no such transaction.

- The number of seconds specified by the scan_sleep_increment parameter is added to each wait until the wait time (in seconds) reaches the value specified by the scan_sleep_max parameter.
- For optimum Replication Agent performance, the value of the scan_sleep_increment parameter should be balanced with the average number of operations applied to marked objects in the primary database over a period of time. In general, better performance results from reading more operations in the transaction log during each Log Reader scan. In a primary database that is infrequently updated, increasing the value of the scan_sleep_increment parameter may improve overall Replication Agent performance.
- If the database is continuously being updated, the value of the scan_sleep_increment parameter is not significant to Replication Agent performance.

scan_sleep_max

The maximum number of seconds between transaction log scans.

Default: 60

Value: An integer from 5 to 86400.

Comments:

- The value of the scan_sleep_max parameter is the maximum number of seconds that can occur before the Log Reader component checks the transaction log in the primary database for a transaction to be replicated after a previous scan yields no such transaction.
- For shorter replication latency in an infrequently updated database, Sybase recommends lower number settings for the scan_sleep_max parameter.
- If the database is continuously being updated, the value of the scan_sleep_max parameter is not significant to Replication Agent performance.

skip_ltl_errors

Determines whether Replication Agent ignores LTL error messages returned from the primary Replication Server.

Default: false

Values

true – Replication Agent logs LTL error messages returned from the primary Replication Server and the offending LTL command(s) and continues processing.

false – Replication Agent stops replication when it receives an LTL error message from the primary Replication Server, if the error is unrecoverable.

Comment

- Use of the skip_ltl_errors parameter is intended for troubleshooting only with assistance from Sybase Technical Support.
- Using the skip_ltl_errors parameter incorrectly may cause data inconsistencies between the primary database and the replicate database.

structured_tokens

Determines whether the Replication Agent Log Transfer Interface component uses structured tokens when generating LTL.

Default

false

Values

true – Replication Agent LTI component uses structured tokens when generating LTL output for the primary Replication Server.

false – Replication Agent LTI component does not use structured tokens when generating LTL output for the primary Replication Server.

Comment

Using structured tokens in the LTL output can improve Replication Server performance, particularly when native datatypes in the primary database must be translated by Replication Server.

truncation_interval

Specifies a time interval between automatic truncations of the Replication Agent transaction log.

Note Sybase Replication Agent does not support log truncation for DB2 Universal Database.

Default

0

Value

An integer from 0 to 720.

Comments

- The value of the truncation_interval parameter is the number of minutes between automatic transaction log truncations.

- Automatic transaction log truncation based on the value of the truncation_interval parameter takes place only when the value of the truncation_type parameter is interval.
- The maximum truncation interval is 720 minutes, or 12 hours.
- If the value of the truncation_interval parameter is 0 (zero) and the value of the truncation_type parameter is interval (the default values for both parameters), automatic truncation is disabled.
- To truncate the transaction log manually, use the pdb_truncate_xlog command.

truncation_type

Configures transaction log truncation behavior of the Replication Agent.

Note Sybase Replication Agent does not support log truncation for DB2 Universal Database.

Default

interval

Values

command – Replication Agent truncates the transaction log only when the pdb_truncate_xlog command is invoked.

interval – Replication Agent truncates the transaction log automatically when determined by a configurable interval of time.

locator_update – Replication Agent truncates the transaction log automatically whenever it receives a new LTM Locator value from the primary Replication Server.

Comments

- Regardless of the value of the truncation_type parameter, the Replication Agent transaction log can be truncated manually at any time by invoking the pdb_truncate_xlog command.
- When the value of the truncation_type parameter is command, the only way the transaction log can be truncated is by invoking the pdb_truncate_xlog command. No automatic truncation takes place when the value of the truncation_type parameter is command.
- When the value of the truncation_type parameter is interval, the transaction log will be truncated automatically when the time interval specified in the truncation_interval parameter has passed.

- When the value of the truncation_type parameter is locator_update, the transaction log will be truncated automatically when Replication Agent receives a new LTM Locator from the primary Replication Server.
- If the value of the truncation_interval parameter is 0 (zero) and the value of the truncation_type parameter is interval (the default values for both parameters), automatic truncation is disabled.
- Replication Agent receives a new LTM Locator based on the values of the lti_update_trunc_point and lr_update_trunc_point properties.

use_rssd

Determines whether Replication Agent accesses the RSSD of the primary Replication Server to get replication definitions.

Default: true

Values: true – Replication Agent accesses the primary Replication Server RSSD to read replication definitions automatically when it goes from the *Admin* state to the *Replicating* state.

false – Replication Agent does not access the primary Replication Server RSSD for replication definitions.

Comments

- If the value of the use_rssd parameter is true, Replication Agent connects to the RSSD of the primary Replication Server to retrieve replication definitions for the primary database whenever the resume command is invoked.
  - Each time it retrieves replication definitions, Replication Agent stores the information in a cache. Replication Agent uses the replication definitions stored in its cache when generating LTL to send to the primary Replication Server.
  - If the Log Transfer Interface (LTI) component encounters an operation for a database object that it does not have cached information for, Replication Agent reconnects to the RSSD to update its replication definition cache.
  - If a replication definition still cannot be found for the operation, the Replication Agent instance suspends operation and goes to the *Admin* state.

- To replicate Unicode data, you must specify the appropriate Unicode datatype in a replication definition, and you must set the value of the use_rssd parameter to true.
- Replication Agent uses the information in replication definitions stored in the RSSD of the primary Replication Server to generate more efficient LTL output, and thus improve throughput in the Log Transfer Interface (LTI) component and throughout the replication system.

  Accessing replication definitions in the RSSD enables the LTI component to improve performance by:

  - Filtering column names from LTL.

    When columns are sent in the order specified in the replication definition, column images can be sent without column names, eliminating some overhead from the LTL.

  - Filtering unneeded columns from LTL.

    When columns are sent as specified in the replication definition, column images for unchanged columns need not be sent, eliminating some overhead from the LTL.

  - Sending data for each column in the datatype specified by the replication definition.

    This allows data to be handled more efficiently all the way through the replication system.

  - Sending database object names in the same character case defined in the replication definitions.
- If the value of the use_rssd parameter is false, none of the previously described performance improvements are possible.
- If the value of the use_rssd parameter is false, all data goes to the primary Replication Server as char datatype.
- If you want to use owner-qualified table names for either primary tables or replicate tables, you must set the value of the use_rssd parameter to true.

Changing Replication Agent configuration parameters

To view, set, or change the current value of a Replication Agent configuration parameter, use the ra_config command.

To change the current Replication Agent administrative user ID (ltm_admin_user) or administrative user password (ltm_admin_pw), use the ra_set_login command. These parameters cannot be changed with the ra_config command and they are not visible in the returned output from that command.

Note Because Replication Agent overwrites the entire configuration file every time the ra_config or ra_set_login command is invoked, and the configuration file contains encrypted passwords, Sybase recommends that you do not edit the configuration file.

For more information on viewing, setting, or changing Replication Agent configuration parameters with the ra_config or ra_set_login command, see Chapter 4, “Sybase Replication Agent Command Reference”.

Replication Agent configuration file

Each Replication Agent instance has its own configuration file. The configuration file is an ASCII file and is named for the Replication Agent instance.

The Replication Agent configuration file is located in a subdirectory with the same name as the Replication Agent instance, within the Replication Agent base directory.

For example, if the Replication Agent base directory is named *rax-12_5* and the Replication Agent instance is named “NY2_RA,” the configuration file *NY2_RA.cfg* is located in the *rax-12_5\NY2_RA* directory. If the Replication Agent software was installed in the default directory on a Windows NT machine, the full path of the configuration file might be:

```
c:\sybase\rax-12_5\NY2_RA\NY2_RA.cfg
```

Configuration file format

The first two lines in the configuration file identify the file as a Replication Agent configuration file and record the time that the file was last modified.

For example:

```
#RA Property File
 #Tue Feb 12 22:41:21 GMT+00:00 2002
```

Each configuration parameter appears on a separate line, followed by an equal symbol (=) and the current value of the parameter.

For example:

```
compress_ltl_syntax=false
```

If the Replication Agent instance is not running, you can view the configuration file to examine the current Replication Agent configuration.

If the Replication Agent instance is running, you can use the ra_config command to examine the current Replication Agent configuration.

For more information on viewing, setting, or changing Replication Agent configuration parameters with the ra_config or ra_set_login command, see Chapter 4, “Sybase Replication Agent Command Reference.”

Note Because Replication Agent overwrites the entire configuration file every time the ra_config or ra_set_login command is invoked, and the configuration file contains encrypted passwords, Sybase recommends that you do not edit the configuration file.

CHAPTER 6

Troubleshooting Sybase Replication Agent

This chapter describes basic troubleshooting procedures for Sybase Replication Agent and the replication system.

This chapter includes the following sections:

- Basic Replication Agent troubleshooting
- Basic Replication Server troubleshooting
- Replication failure troubleshooting tips

Basic Replication Agent troubleshooting

This section contains basic procedures you can use to check the operation of Sybase Replication Agent.

The following topics are in this section:

- Problems with resolving server names
- Problems with marking tables and stored procedures
- Problems with unmarking tables
- Problems with changed or missing replication definitions
- Problems with non-standard datatypes in primary key columns
- Problems with smallint and tinyint datatypes
- Problems with file handles
- Recovering from a lost configuration file

Problems with resolving server names

If you experience problems connecting Sybase Replication Agent with other servers in a replication system (the primary database server, Replication Server, or the RSSD) because the server name cannot be resolved, try using the actual IP address of the server.

Problems with marking tables and stored procedures

Note This section is not applicable to Sybase Replication Agent support for DB2 Universal Database.

When a primary table is marked for replication, trigger-based Replication Agents create a shadow table to record the replicated operations. The shadow table has all the columns of the primary table, plus three or four columns (depending on the type of primary database) that the Replication Agent creates for its use:

- ra_tran_id_
- ra_img_type_
- ra_opid_ (Microsoft SQL Server and Oracle only)
- ra_opid_hi (Informix only)
- ra_opid_lo (Informix only)

These names are fixed and cannot be changed.

Note In an Informix primary database, column names in a primary table may also conflict with global variable names that the Replication Agent uses in its stored procedures. See Appendix B, “Administering the Replication Agent for Informix,” for more information.

When a stored procedure is marked for replication, the Replication Agent creates a shadow table with a column for each parameter of the stored procedure, plus the columns the Replication Agent needs for its own use.

If the name of one of the columns in the primary table (or one of the parameters of the stored procedure) is the same as one of the Replication Agent columns in the shadow table, a SQL execution error occurs when you attempt to mark the table (or stored procedure). In addition:

- The pdb_setreptable or pdb_setrepproc command returns an error.
- The Replication Agent system log records the duplicate column name error message.
- The marking script is generated, but not executed for the primary table (*%SYBASE%\rax-12_5\instname\tablename\mark.sql*) or stored procedure (*%SYBASE%\rax-12_5\instname\procname\mark.sql*).
- All occurrences of the offending object name in the marking script are delimited as: <<name>>

To solve this problem, you have two options:

- Change the name of the offending column or parameter in the primary object, then use the pdb_setreptable or pdb_setrepproc command again to mark the object.
- If you want to keep the current name of the offending column or parameter in the primary object, you can edit the marking script (*mark.sql*) to change the name of the offending column (or parameter) in both shadow table DDL commands and the shadow row procedures, and remove the error delimiters (<< and >>) and leave the original column (or parameter) name unchanged in trigger code. After editing the marking script, you must execute it manually in the primary database.

Whether you change the primary object names in the database or modify the marking script, the Replication Agent replicates data using the new names, and you must make some accommodation at the replicate database to the name changes at the primary database.

You can either change the column (or parameter) names of the tables (or procedures) in the replicate database, or create a Replication Server function string to map the new primary object names to the original names in the replicate database.

Problems with unmarking tables

When you unmark an object that was marked for replication, Replication Agent generates a script that removes from the primary database the transaction log objects that were created to facilitate replication for that object.

If the marked object (table or stored procedure) is dropped from the database before it is unmarked by Replication Agent, the unmark process will fail because the unmark script will not find all the transaction log components to be removed.

To solve this problem, you can use the unmark, force options with the pdb_setreptable or pdb_setrepproc commands. The force option will cause the SQL script to continue execution, even if it encounters errors, thus removing any existing transaction log objects, even though some are missing.

Problems with cascading deletes

Note This section is not applicable to Sybase Replication Agent support for DB2 Universal Database.

To capture data for replication, the trigger-based Sybase Replication Agent creates insert, update, and delete triggers for each table to be replicated. The triggers are created during the table-marking process.

Child tables subject to being “cascade deleted” when their parent table is deleted cannot have a delete trigger defined. The same problem occurs if you define a cascade delete rule for a table that has a delete trigger defined. Some database servers do not allow cascade delete rules for tables with a delete trigger defined.

The JDBC protocol includes many cascade delete RI rules. To use a trigger-based Sybase Replication Agent, these rules must be dropped for all tables marked for replication.

If cascade delete operations are required in a primary database, then:

- The applications (managers) need to delete the child tables first, or
- Additional logic must be built into the parent tables' delete triggers to delete the child tables.

Problems with changed or missing replication definitions

If the value of the use_rssd configuration parameter is true, Replication Agent connects to the RSSD of the primary Replication Server to retrieve replication definitions for the primary database whenever the resume command is invoked. Each time it retrieves replication definitions, Replication Agent stores that information in a cache. Replication Agent uses the replication definitions stored in its cache when generating LTL to send to the primary Replication Server.

Note In this section, the term *replication definition* refers to both table replication definitions and function replication definitions.

If the Log Transfer Interface (LTI) component encounters an operation for a database object that it does not have cached information for, Replication Agent reconnects to the RSSD to update its replication definition cache. If a replication definition still cannot be found for the operation, the Replication Agent instance suspends operation and goes to the *Admin* state.

If you drop a replication definition after Replication Agent has cached the RSSD information, one of two problems may occur:

- If the replication definition is dropped and then re-created, the primary Replication Server may suspend the Replication Agent connection when it receives LTL that is not formatted correctly for the object, as specified in the replication definition.
- If the replication definition is dropped and not replaced (that is, it no longer exists), the primary Replication Server will simply ignore the LTL it receives for an object that is not specified in a replication definition.

The more difficult problem is the second one, because there is no failure indication other than replication not taking place as expected. In that case, neither Replication Server nor Replication Agent are aware of the problem, and hence, neither will log it as an error.

To avoid this problem, you should quiesce the Replication Agent instance, change the replication definition, then resume the Replication Agent instance so the Replication Agent has the correct version of the replication definition in its cache.

Problems with non-standard datatypes in primary key columns

Sybase Replication Agent recognizes all the standard native datatypes used in the supported primary database servers. However, some database servers allow you to create “user-defined datatypes” in the primary database. When the Replication Agent encounters a non-standard datatype in the primary database, it does not recognize the data in that column, and it does not capture any data from that column when it records a transaction.

The primary key column(s) are used to generate a where clause in the transaction operations applied at the replicate database. If no data is sent for a

primary key column, the SQL executed at the replicate database will fail because the where clause will be empty (or contain incomplete data).

To avoid this problem, do not identify a column that contains a non-standard datatype as a primary key column in a replication definition.

Problems with smallint and tinyint datatypes

A problem exists with earlier versions of Replication Server (prior to version 12.0) where Replication Server fails to properly convert character datatypes to smallint and tinyint datatypes.

There are two ways to avoid this problem with earlier versions of Replication Server:

- Set the value of the Replication Agent use_rssd configuration parameter to true.
- Use the int datatype rather than the smallint or tinyint datatypes in replication definitions.

Problems with file handles

A connection failure error may occur when there is an insufficient number of file handles available to Replication Agent.

The following error message may indicate this problem:

```
java.lang.NoClassDefFoundError: ... ConnectFailedEvent
at ... FailedToConnect
```

If this error occurs while Replication Agent is otherwise functioning normally (that is, the entire replication system seems to be working properly) and you cannot find another cause for the connection failure, it may indicate a file handles problem.

To solve this problem, increase the number of file handles available to Replication Agent.

Recovering from a lost configuration file

If the Replication Agent configuration file is deleted or damaged, the Replication Agent instance cannot initialize correctly upon start-up and will immediately shut itself down.

When the Replication Agent configuration file is deleted or damaged, you have two options for recovery:

- Copy the default configuration file to the Replication Agent instance directory and then re-create the missing configuration items.
- Delete the Replication Agent instance, create a new instance, and then go through the setup process to re-create the configuration.

Note The information you recorded on the "Sybase Replication Agent Installation Worksheet" can be used to re-create a damaged or missing Replication Agent configuration file.

Use the following procedure to recover from a damaged or missing Replication Agent configuration file by copying the default configuration and re-creating the missing items

❖ **To recover from a lost or missing Replication Agent configuration file**

1 Make sure the Replication Agent instance is shut down. If it is not, use the shutdown command to terminate the Replication Agent instance.

2 Copy the default Replication Agent configuration file (such as *rai.cfg*) from the *rax-12_5\config* directory to the Replication Agent instance directory.

For example, if the Replication Agent instance is named "NY2_RA," you would use the following command at the operating system prompt in the Replication Agent base directory to copy the default Replication Agent for Oracle configuration file:

```
copy config\rao.cfg NY2_RA\NY2_RA.cfg
```

3 Using your the access tool for your primary database (such as Informix dbaccess), log in to the primary database and find the Replication Agent transaction log system table *ra*_xlog_system_, where *ra_* is the prefix string you specified in the pdb_xlog_prefix parameter. Once you find the transaction log system table, make a note of the prefix string so you can reset the value of the pdb_xlog_prefix parameter.

4 Start the Replication Agent instance using the following command at the operating system prompt:

```
ra -i inst_name
```

where *inst_name* is the name of the Replication Agent instance you want to start.

5 Log in to the Replication Agent administration port using the following command at the operating system prompt:

```
isql -Usa -P -Sinst_name
```

where *inst_name* is the name of the Replication Agent instance you want to administer.

6 Use the ra_config command to set the log directory for the Replication Agent instance:

```
ra_config log_directory, logpath
```

where *logpath* is the full path specification for the Replication Agent instance.

The following example is the path specified automatically when the Replication Agent instance is created:

```
C:\rax-12_5\inst_name\log\
```

where *inst_name* is the name of the Replication Agent instance.

7 Use the ra_set_login command to reset the administrative user ID and password for the Replication Agent instance:

```
ra_set_login username, password
```

where *username* is the user ID of the Replication Agent instance administrative user and *password* is the corresponding password.

8 Use the ra_config command to set the values of the following Replication Agent configuration parameters for the primary database:

pdb_xlog_prefix
pds_charset
pds_database_name
pds_datasource_name
pds_host_name
pds_password
pds_port_number
pds_server_name

pds_service_name
pds_username

9 Use the ra_config command to set the values of the following Replication Agent configuration parameters for the primary Replication Server:

rs_host_name
rs_password
rs_port_number
rs_source_db
rs_source_ds
rs_username

10 Use the ra_config command to set the values of the following Replication Agent configuration parameters for the RSSD of the primary Replication Server:

rssd_charset
rssd_database_name
rssd_host_name
rssd_password
rssd_port_number
rssd_username

11 Use the shutdown command to shut down the Replication Agent instance so you can re-start the Replication Agent instance with the correct administrative user ID:

```
shutdown
```

12 Start the Replication Agent instance using the following command at the operating system prompt:

```
ra -i inst_name
```

where *inst_name* is the name of the Replication Agent instance you want to start.

13 Log in to the Replication Agent administration port using the following command at the operating system prompt:

```
isql -Usa -P -Sinst_name
```

where *inst_name* is the name of the Replication Agent instance.

14 Use the ra_status command to verify that the Replication Agent instance is in the *Admin* state:

```
ra_status
```

15 Use the test_connection command to verify that the Replication Agent instance can communicate with the primary database server, the primary Replication Server, and the RSSD of the primary Replication Server:

```
test_connection
```

16 Use the resume command to put the Replication Agent instance into the *Replicating* state.

```
resume
```

17 Use the ra_status command to verify that the Replication Agent instance is in the *Replicating* state:

```
ra_status
```

If the Replication Agent instance successfully enters the *Replicating* state, then the information stored in the configuration file is correct for that Replication Agent instance.

If the Replication Agent instance remained in the *Admin* state after you invoked the resume command, then you still need to verify and correct the following items:

- The prefix string recorded in the Replication Agent pdb_xlog_prefix parameter.
- Configuration parameters for the primary database (see step 8 in the previous procedure).
- Configuration parameters for the primary Replication Server (see step 9 in the previous procedure).
- Configuration parameters for the RSSD of the primary Replication Server (see step 10 in the previous procedure).

Basic Replication Server troubleshooting

This section contains basic procedures you can use to check the operation of the Replication Server(s) in your system.

See Also

- Replication Server *Troubleshooting Guide*

Verify whether Replication Server is running

Verify whether your primary and intermediate Replication Servers are running. Use the Replication Server admin who is up command to find out which servers are running. Use the Replication Server admin who is down command to find out which servers are down.

Check the Replication Server log

The Replication Server log contains error and warning messages. The most recent messages are at the end of the log.

Display information about stable queues

You can check the Replication Server queues to determine which transactions are being processed or ignored, and to determine whether transactions are open, or not being committed.

❖ **To display information about SQM and SQT threads**

1 Log in to the primary Replication Server and use the admin who, sqm command.

2 View the results to determine the number of duplicate messages being detected and ignored, and the number of blocks being written in the Replication Server stable queues.

3 In the primary Replication Server, use the admin who, sqt command.

4 View the results to find open transactions.

See Also

- Replication Server *Reference Manual* for information about the admin who commands and results.

Correct the origin queue ID

If insert or update operations in the primary database are not being replicated, and if you do not find any errors in the Replication Agent system log or trace log, the most likely problem is that the values of the origin queue ID (*qid*) sent to the primary Replication Server are lower than previous values of the *qid*, and therefore, the new operations are being ignored by the primary Replication Server.

❖ **To correct the problem of incorrect *qid* values**

1 Turn on tracing for the LTITRACELTL trace flag.

You can view Replication Agent LTL output by examining the contents of the *LTITRACELTL.log* file.

2 Log in to the primary Replication Server and use the admin who, sqm command to determine if any duplicate operations are in the inbound queue of the primary Replication Server.

If you find duplicate operations in the inbound queue of the primary Replication Server, then Replication Agent is re-sending operations to the primary Replication Server as part of a recovery process. Finding duplicate operations indicates that the value of the Replication Agent origin queue ID is not the problem.

If you find no duplicate operations in the inbound queue of the primary Replication Server, continue this procedure to reset the *qid* value.

3 Log in to the Replication Agent administration port using the following command:

```
isql -Uusername -Ppassword -Sinst_name
```

where *username* is the Replication Agent administrative user ID, *password* is the corresponding password, and *inst_name* is the name of the Replication Agent instance.

4 Use the Replication Agent quiesce command to bring the Replication Agent instance to the *Admin* state.

```
quiesce
```

5 Use the Replication Agent ra_locator command to set the value of the LTM Locator stored by the Replication Agent instance to all zeros.

```
ra_locator zero
```

6 Log in to the RSSD of the primary Replication Server and execute the rs_zeroltm stored procedure.

```
rs_zeroltm PDS, PDB
```

where *PDS* is the name of the primary data server and *PDB* is the name of the primary database, as specified in the create connection command in the primary Replication Server.

7 Restart replication by using the Replication Agent resume command.

```
resume
```

After the value of the LTM Locator is set to all zeros in both the Replication Agent transaction log and the RSSD of the primary Replication Server, Replication Agent starts scanning the transaction log.

Trigger-based Replication Agents (for Informix, Microsoft SQL Server, and Oracle) start scanning the transaction log from the beginning of the log. All operations in the transaction log are sent to the inbound queue of the primary Replication Server.

Log-based Replication Agents (for DB2 Universal Database) start scanning the transaction log at the end of the log. Any previous transactions are not sent, including transactions that may not have been sent before. In that event, a loss of data may occur. For more information on transaction log positioning, see "Handling repositioning in the log" on page 207.

Replication failure troubleshooting tips

Use the procedures in this section to diagnose the source of replication failure.

Verify names in Replication Server

Check the Replication Server log to verify that the values for *data_server* and *database* specified in the create connection command exactly match the values specified for the Replication Server RS_source_db and RS_source_ds configuration parameters and that they exactly match the values specified for the Replication Agent rs_source_db and rs_source_ds configuration parameters.

Use the Replication Server admin who command to look for the Replication Agent instance in the list of replication system servers.

Verify the existence of rs_username

Executing the Replication Server connect source lti command accomplishes three things:

- Verifies that the database connection from the Replication Agent instance to the primary Replication Server exists.

- Verifies that the user ID specified in the Replication Agent rs_username parameter has permission to connect to the primary Replication Server as a data source.
- Returns a version string that displays the highest numbered version of LTL that the primary Replication Server supports.

❖ **To verify that the user ID specified in the *rs_username* parameter exists and has appropriate permissions in the primary Replication Server**

1 Log in to the primary Replication Server as the user name you specified as the value of the rs_username configuration parameter (see the "Sybase Replication Agent Installation Worksheet" for this value).

2 Execute the connect source lti command using the following example syntax:

```
connect source lti data_server.database version
```

where *data_server* is the value you specified for the Replication Agent rs_source_ds configuration parameter, *database* is the value you specified for the Replication Agent rs_source_db configuration parameter, and *version* is the proposed LTL version number.

Refer to the installation and setup worksheet in the Sybase Replication Agent *Installation Guide* for the values of the rs_source_ds and rs_source_db parameters.

For the value of the LTL version number, you can enter `999` and Replication Server will return the highest numbered version of LTL that it supports.

3 Disconnect from the primary Replication Server as rs_username before you invoke the Replication Agent resume command.

See Also

- Replication Server *Design Guide* for more information about the connect source lti command.

Check the Replication Agent logs

The Replication Agent system log and trace log files contain warning and error messages, as well as information about the connections to Replication Server and the primary database.

Check the replicate database

Issue the admin who_is_down command to determine whether any connection(s) to the replicate database(s) is down, or has been suspended by Replication Server.

Check replication definitions and subscriptions

Verify that you created replication definitions with the appropriate information.

Verify that you defined and activated subscriptions for all the replication definitions.

Check the ASE error log

The ASE error logs contain warning and error messages about the Adaptive Server and RSSD.

Verify whether maintenance user ID owns transactions

If the value of the filter_maint_userid parameter is set to true, Replication Agent does not process changes made by the Replication Server maintenance user ID in the primary database.

APPENDIX A

Administering the Replication Agent for UDB

The term “Replication Agent for UDB” refers to an instance of the Sybase Replication Agent version 12.5 software configured for a primary database that resides on an IBM DB2 Universal Database server. This appendix describes the characteristics of the Sybase Replication Agent that are unique to the Replication Agent for UDB implementation.

Note For information on the basic features and operation of Sybase Replication Agent version 12.5, refer to the other parts of this Sybase Replication Agent *Administration Guide*.

This appendix includes the following sections:

- DB2 Universal Database-specific issues
- Replication Agent for UDB transaction log
- Replication Agent for UDB setup test scripts

DB2 Universal Database-specific issues

This section describes general issues and considerations that are specific to using Sybase Replication Agent version 12.5 with the IBM DB2 Universal Database server.

The following topics are included in this section:

- Log-based Replication Agent
- Replication Agent connectivity
- DB2 Universal Database Administration Client
- Database logging issues
- Handling repositioning in the log

- Replication Agent behavior
- Character case of database object names
- Format of origin queue ID
- Datatype compatibility

Log-based Replication Agent

The Replication Agent for UDB is a log-based Replication Agent, unlike the Sybase Replication Agent implementations for Informix, Microsoft SQL Server, and Oracle database servers.

Because of its log-based implementation, and because of differences in the behavior of the IBM DB2 Universal Database server compared to other database servers supported by Sybase Replication Agent, the Replication Agent for UDB uses different methods to acquire transaction information from the primary database. In addition, a few of the common Sybase Replication Agent features either work differently or are not available with the Replication Agent for UDB.

Feature differences in Replication Agent for UDB

The following Sybase Replication Agent features have unique behavior in the Replication Agent for UDB:

- Creating a transaction log
- Marking a table for replication
- Support for rs_marker and rs_dump function calls

Creating a transaction log

The Replication Agent for UDB provides the same features for creating and removing the “transaction log” as other implementations of the Sybase Replication Agent. However, since the actual transaction log used by Replication Agent for UDB is the native transaction log of the DB2 Universal Database server, Replication Agent for UDB creates fewer tables in the primary database to store its system information. The Replication Agent for UDB does not create any stored procedures or triggers in the primary database.

Because the Replication Agent for UDB requires access to the DB2 Universal Database transaction log, the user ID that the Replication Agent uses to access the primary database must have either SYSADM or DBADM authority in the

database. This user ID is stored in the Replication Agent pds_username configuration parameter.

Note If the user ID specified in the pds_username parameter does not have either SYSADM or DBADM authority in the primary database, the pdb_xlog create command returns an error.

For more information about the Replication Agent for UDB transaction log, see "Replication Agent for UDB transaction log" on page 213.

Marking a table for replication

The Replication Agent for UDB provides the same features for marking and unmarking tables for replication as other implementations of the Sybase Replication Agent. However, the Replication Agent for UDB does *not* create any stored procedures or triggers in the primary database.

When marking a table for replication, Replication Agent for UDB alters the table to set the DATA CAPTURE attribute to DATA CAPTURE CHANGES. When the table is unmarked, the table is altered to return to its original DATA CAPTURE attribute.

Note Do not change the DATA CAPTURE attribute of a table marked for replication with the Replication Agent for UDB.

Support for rs_marker function calls

The Replication Agent for UDB does not provide a stored procedure for the Replication Server rs_marker function call because the DB2 Universal Database server does not support procedures stored in the database. The Replication Agent for UDB provides shadow tables to support Replication Server rs_marker function calls.

The rs_marker function is used only if Replication Server is configured to perform automatic materialization.

To set up automatic materialization for the DB2 Universal Database server, you must create an rs_marker custom function string in Replication Server.

Before you create the rs_marker function string, you must use the Replication Agent pdb_xlog command to create the Replication Agent transaction log system tables in the primary database. You can then invoke the pdb_xlog command with no parameters to determine the actual name of the shadow table associated with rs_marker.

When you create the custom function string for rs_marker, you must use an insert statement that refers to the actual shadow table generated by the Replication Agent pdb_xlog command.

Features not available in Replication Agent for UDB

The following Sybase Replication Agent features are not available with the Replication Agent for UDB:

- Stored procedure replication
- Transaction log truncation

Note When Replication Agent commands related to these features are invoked, they return an error.

Stored procedure replication

Stored procedure replication is not available with the Replication Agent for UDB. Therefore, the pdb_setrepproc command is not supported in Replication Agent for UDB.

Transaction log truncation

Because the Replication Agent for UDB uses the native database transaction log of the DB2 Universal Database server, it cannot control log truncation as it does in other implementations of the Sybase Replication Agent. Therefore, the pdb_truncate_xlog command is not supported in Replication Agent for UDB.

The following log truncation-related configuration parameters have no effect in Replication Agent for UDB:

- truncation_interval
- truncation_type

Note For information about archiving the DB2 Universal Database transaction log, see “Archiving the transaction log” on page 216.

DB2 Universal Database Administration Client

If the Replication Agent for UDB software is installed on a different host machine from the DB2 Universal Database server, you must install the DB2 Universal Database Administration Client on the same host machine as the Replication Agent.

If the Replication Agent for UDB software is installed on the same host machine as the DB2 Universal Database server, a separate DB2 Universal Database Administration Client is not required.

In either case, you must configure an ODBC data source in the DB2 Universal Database Administration Client, then use the database name and database alias

specified for that ODBC data source when you configure Replication Agent for UDB connectivity.

Note When you install the DB2 Universal Database Administration Client software on a Windows NT or Windows 2000 host, an environment variable named *LC_ALL* is created. This environment variable may conflict with certain Sybase utilities, such as isql and dsEdit. If a conflict occurs, you can simply remove the *LC_ALL* environment variable.

To configure the DB2 Universal Database Administration Client for the Java version required by Sybase Replication Agent 12.5, you must run the *usejdbc2* script in the *path_name/sqllib/java12* directory, where *path_name* is the path where you installed the DB2 client.

Replication Agent connectivity

Connectivity between the Replication Agent for UDB and the DB2 Universal Database server is by way of the DB2 Universal Database JDBC driver.

The DB2 Universal Database JDBC driver is installed on the Replication Agent host machine when you install the DB2 Universal Database Administration Client software. You must include the path of the JDBC driver in the *CLASSPATH* environment variable before you start the Replication Agent for UDB.

For more information on configuring the DB2 Universal Database JDBC driver, see the Sybase Replication Agent *Installation Guide*.

Configuration parameters

The following Replication Agent configuration parameters are required to configure a connection between the Replication Agent for UDB and a DB2 Universal Database server:

- pds_username – must have SYSADM or DBADM authority
- pds_password – for user ID specified in pds_username
- pds_database_name – database name
- pds_datasource_name – database alias
- pds_connection_type – UDBODBC (literal value)

Database logging issues

There are two database logging issues specific to DB2 Universal Database:

- DB2 Universal Database LOGRETAIN parameter
- Replication Agent read buffer size

The LOGRETAIN parameter

The Replication Agent for UDB requires the DB2 Universal Database LOGRETAIN parameter be set to RECOVERY MODE for the primary database.

Note This also requires archive logging, not circular logging.

Read buffer size

The Replication Agent for UDB LogReader component uses the value of the max_ops_per_scan parameter to determine the maximum number of bytes to be read from the transaction log during each scan. Because the LogReader reads bytes, it requires a buffer to store the bytes read.

The LogReader component determines the maximum size of this buffer by multiplying the value of the max_ops_per_scan parameter by 10. For example, if the value of the max_ops_per_scan parameter is 1000 (the default), the size of the LogReader read buffer is 10,000 bytes.

It is very difficult to identify a minimum buffer size that will always work. The value range of max_ops_per_scan is 25 to 2,147,483,647, which means the smallest size of the buffer is 250 bytes.

If the read buffer size is too small to read one operation, LogReader reports an error and shuts down the Replication Agent instance.

The error Replication Agent passes from the database server is -30081. Unfortunately, this is a general communication error that DB2 Universal Database uses to report various communication problems, not just for an insufficient buffer size.

Handling repositioning in the log

The Replication Agent uses the value of the LTM locator received from the primary Replication Server to determine where it should begin looking in the transaction log for transactions to be sent to the Replication Server.

The Replication Agent for UDB uses the LTM locator value as follows:

- When the value of the LTM locator received from Replication Server and the LTM locator stored by Replication Agent are both zero (0), the Replication Agent positions the Log Reader component at the end of the DB2 Universal Database transaction log.
- When the value of the LTM locator received from Replication Server and the LTM locator stored by Replication Agent are both not zero, Replication Agent uses the LTM locator value it received from Replication Server to determine the starting position of the oldest open transaction and positions the Log Reader component at that location in the DB2 Universal Database transaction log.
- When the value of the LTM locator received from Replication Server is zero (0) and the value of the LTM locator stored by Replication Agent is not zero, Replication Agent uses the LTM locator value it has stored to determine the starting position of the oldest open transaction and positions the Log Reader component at that location in the DB2 Universal Database transaction log.

Warning! In the event that both LTM locator values are zero, there is a remote possibility of data loss. Two specific conditions could cause this data loss:

- Repositioning the Log Reader at the end of the transaction log may cause data loss if there are replicated transactions that have not been processed at the time the Log Reader is repositioned.
- When the Replication Agent Log Reader component goes to the *Replicating* state, it does so asynchronously. When you receive a prompt after invoking the resume command, the Log Reader component may not be finished getting into the *Replicating* state and positioning itself at the end of the log. If you mark a table immediately after the prompt returns from the resume command, it is possible that the record containing the mark information will be written to the log before the Log Reader component has positioned itself. In that case, the Log Reader component will miss that record and not replicate any subsequent data for that table. To avoid this problem, you should wait a short time after invoking the resume command before marking a table for replication.

Replication Agent behavior

The following Replication Agent issues are unique to Replication Agent for UDB:

- Marking tables immediately after going to *Replicating* state, when the value of the LTM locator is 0 (zero).
- Forcing applications off the primary database with the DB2 FORCE APPLICATION command.

Marking tables immediately after resume when LTM locator is zero

When the Replication Agent instance goes to Replicating state, the Log Reader component reads the primary database transaction log and uses the value of the origin queue ID to determine the position in the log to start reading. When the value of the LTM locator is 0 (zero), the Log Reader starts reading at the end of the log.

Because the Log Reader's operation is asynchronous, the Replication Agent instance can return to the operating system prompt after the resume command, but before the Log Reader has completed its start-up process. If you immediately invoke the pdb_setreptable command to mark a table for replication after the resume command returns, the mark object entry can be placed in the transaction log before the Log Reader finds the end of the log. In that event, the Log Reader will miss the mark table entry and table marking will fail.

To avoid this problem, wait five to ten seconds after invoking the resume command before invoking the pdb_setreptable command to mark a table.

Forcing applications off the database

The DB2 FORCE APPLICATION command causes the database server to drop its connections with an application. The FORCE APPLICATION ALL command causes the database server to drop its connections with all applications.

If you invoke the FORCE APPLICATION command and specify either the Replication Agent application handle or the ALL keyword, the database server drops its connections with the Replication Agent instance. In that event, the Replication Agent receives DB2 error code -30081 and cannot recover, so the Replication Agent instance shuts itself down.

To avoid this situation, invoke the Replication Agent quiesce command before using the DB2 FORCE APPLICATION command.

Character case of database object names

Database object names must be delivered to the primary Replication Server in the same format as they are specified in replication definitions. For example, if a replication definition specifies a table name in all uppercase, then that table name must appear in all uppercase when it is sent to the primary Replication Server by the Replication Agent.

Note Replication will fail if a database object name is delivered to the primary Replication Server in a format different from that specified in the replication definition.

Sybase Replication Agent version 12.5 gives you some control over the way it treats the character case of database object names when it sends LTL to the primary Replication Server. You have three options:

- as-is — database object names are passed to Replication Server in the same format as they are actually stored in the primary database server
- lower — database object names are passed to Replication Server in *all lowercase*, regardless of the way they are actually stored in the primary database server
- upper — database object names are passed to Replication Server in *all uppercase*, regardless of the way they are actually stored in the primary database server

You specify the character case option you want by setting the value of the ltl_character_case configuration parameter. The parameter values are asis, lower, and upper. The default value of the ltl_character_case parameter is asis.

In the case of the IBM DB2 Universal Database server, database object names are stored in all uppercase. Therefore, the asis and upper options have the same affect on database object names sent from Replication Agent for UDB to the primary Replication Server.

Format of origin queue ID

Each record in the transaction log is identified by an origin queue ID that consists of 64 hexadecimal characters (32 bytes). The format of the origin queue ID is determined by the Replication Agent instance, and it varies according to the primary database type.

Table A-1 illustrates the format of the origin queue ID for the Replication Agent for UDB.

Table A-1: Replication Agent for UDB origin queue ID

Character	Bytes	Description
0-3	2	database generation ID
4-19	8	operation sequence number
20-35	8	transaction ID
36-51	8	first operation sequence number of oldest active transaction
52-55	2	operation type (begin = 0, data/LOB = 1, commit/rollback = 7FFF)
56-59	2	LOB sequence ID
60-63	2	unused

Datatype compatibility

Replication Agent for UDB processes transactions and passes data to the primary Replication Server.

The primary Replication Server uses the datatype formats specified in the replication definition to receive the data from Replication Agent for UDB.

The following table describes the default conversion of DB2 Universal Database datatypes to Sybase datatypes.

Table A-2: DB2 Universal Database to Sybase default datatype mapping

DB2 UDB datatype	DB2 UDB length/range	Sybase datatype	Sybase length/range	Notes
BIGINT	-9,223,372,036, 854,775,808 to 9,223,372,036, 854,775,807.	decimal	10^{-38} to 10^{38}, 38 significant digits.	
BLOB	variable length, 2GB, binary data	image	2GB	
CHAR	254 bytes	char	32K	
CHAR FOR BIT DATA	254 bytes, binary data	binary	32K	
CLOB	variable length, 2GB, character data	text	2GB	

DB2 UDB datatype	DB2 UDB length/range	Sybase datatype	Sybase length/range	Notes
DATE	0001-01-01 to 9999-12-31	char or datetime	32K (char)	If the pdb_convert_datetime parameter is false, DATE values are sent as char datatype strings. If the pdb_convert_datetime parameter is true, DATE values are converted to datetime values.
DBCLOB	variable length, 2GB, double-byte character data	text	2GB	
DECIMAL	$-10^{31}+1$ to $10^{31}-1$, 31 digits of precision	decimal	10^{-38} to 10^{38}, 38 significant digits.	
DOUBLE				See FLOAT.
FLOAT	8 bytes, -1.79769^{308} to 1.79769^{308}	float	The float precision and range corresponds to a C double datatype, approximately 16 significant digits.	Extremely small values are truncated to 16 digits to the right of the decimal. Extremely large values retain their precision.
GRAPHIC	127 characters, double-byte character data	unichar or char	32K	To use the unichar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
INTEGER	-2,147,483,648 to 2,147,483,647	int	-2,147,483,648 to 2,147,483,647.	
LONG VARCHAR	variable length, 32,700 bytes, character data	varchar	32K	
LONG VARCHAR FOR BIT DATA	32,700 bytes, binary data	varbinary	32K	
LONG VARGRAPHIC	16,350 characters, double-byte character data	univarchar or varchar	32K	To use the univarchar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
NUMERIC (synonym for DECIMAL)				See DECIMAL.
REAL	-3.402^{38} to 3.402^{38}	decimal	10^{-38} to 10^{38}, 38 significant digits.	
SMALLINT	-32,768 to 32,767	smallint	-32,768 to 32,767	

DB2 UDB datatype	DB2 UDB length/range	Sybase datatype	Sybase length/range	Notes
TIME	00:00:00 to 24:00:00	char or datetime	32K (char)	
TIMESTAMP	0001-01-01-00.00.00.000000 to 9999-12-31-24.00.00.000000	char or datetime	32K (char)	If the pdb_convert_datetime parameter is false, TIMESTAMP values are sent as char datatype strings. If the pdb_convert_datetime parameter is true, TIMESTAMP values are converted to datetime values.
VARCHAR	32,672 bytes	varchar	32K	
VARCHAR FOR BIT DATA	32,672 bytes, binary data	varbinary	32K	
VARGRAPHIC	16,336 characters, double-byte character data	univarchar or varchar	32K	To use the univarchar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.

For each datatype in Table A-2, lengths in the second column are described as:

- Character datatypes – maximum number of bytes
- Graphic datatypes – maximum number of characters
- Numeric datatypes – range from smallest to largest values
- Temporal datatypes – range from earliest time to latest time

DB2 Universal Database datatype restrictions

If you use a version of Replication Server prior to version 12.5:

- DB2 VARCHAR, VARCHAR FOR BIT DATA, VARGRAPHIC, and the LONG variants of those dataypes that contain more than 255 bytes will be truncated to 255 bytes.
- DB2 GRAPHIC, LONG VARGRAPHIC, and VARGRAPHIC multi-byte character dataypes will be replicated as char or varchar single-byte datatypes.

Replication Agent for UDB transaction log

The Replication Agent for UDB uses the native database transaction log maintained by the DB2 Universal Database server to capture transactions in the primary database for replication. The Replication Agent for UDB also creates tables in the primary database for its system information.

Note Unlike other implementations of the Sybase Replication Agent, the Replication Agent for UDB does not create stored procedures and triggers in the primary database.

The Replication Agent transaction log is created by invoking the pdb_xlog command with the create keyword. When you invoke this command, Replication Agent generates a SQL script that is run in the primary database. This script (stored in the *create.sql* file in the *rax-12_5\inst_name\scripts\xlog* directory) creates the Replication Agent transaction log components referred to as the *transaction log base objects*. The transaction log base objects must be created before any objects can be marked for replication in the primary database.

Note This section describes the schema and details of the Replication Agent transaction log for a primary database that resides in the DB2 Universal Database server. For more general information about the Replication Agent transaction log, see Chapter 3, "Administering Sybase Replication Agent" in this book.

Transaction log components

There are several tables used for the Replication Agent transaction log in the primary database. All tables of the Replication Agent transaction log contain at least one index, and some contain more than one index.

Transaction log object names

There are two variables in the Replication Agent transaction log table names shown in this appendix:

- *prefix* — represents the one- to three-character string value of the pdb_xlog_prefix parameter (the default is ra_).

- *xxx* — represents an alphanumeric counter, a string of characters that is (or may be) added to a table name to make that name unique in the database.

The value of the pdb_xlog_prefix parameter is the prefix string used in all Replication Agent transaction log table names.

If this value conflicts with the names of existing database objects in your primary database, you can change the value of the pdb_xlog_prefix parameter by using the ra_config command.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log tables in the primary database. If you change the value of pdb_xlog_prefix after you create the Replication Agent transaction log, the Replication Agent instance will not be able to find the transaction log tables that use the old prefix.

You can use the pdb_xlog command to view the names of Replication Agent transaction log tables in the primary database.

Transaction log base tables

Table A-3 lists the Replication Agent system tables that are considered Replication Agent transaction log base objects. No permissions are granted on these tables when they are created.

Table A-3: Replication Agent Transaction Log Base Tables

Table	Database Name
transaction log system table	*prefix*xlog_system_
marked objects table	*prefix*marked_objs_*xxx*
LOB columns table	*prefix*blob_columns_*xxx*
Log Admin work table	*prefix*rawork_*xxx*
proc active table	*prefix*procactive_*xxx*
force record table	*prefix*force_record_*xxx*

Marker shadow tables

Table A-4 lists the marker shadow tables that are considered Replication Agent transaction log base objects.

Table A-4: Replication Agent Marker Shadow Tables

Procedure/Table	Database Name
rs_marker shadow table	*prefix*markersh_*xxx*
rs_dump shadow table	*prefix*dumpsh_*xxx*

Getting actual names of the transaction log objects

The Replication Agent instance generates the names of its transaction log tables. If you want to find out the actual names of transaction log tables, you must use the pdb_xlog command.

❖ **To find out the names of transaction log base tables**

- At the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

  ```
  pdb_xlog
  ```

  The pdb_xlog command returns a list of the transaction log base objects in the primary database.

Marked objects table

One of the Replication Agent transaction log base objects is the **marked objects table**. The marked objects table contains an entry for each marked table in the primary database. Each marked table entry contains the following information:

- the name of the marked primary object (table)
- the primary object's replicated name
- the type of the primary object (table only, in Replication Agent for UDB)
- the "replication enabled" flag for the primary object
- the owner of the primary object
- the "send owner" flag
- the tablespace ID of the primary object
- the table ID of the primary object
- the "convert datetime" flag
- original value of the table's DATA CAPTURE attribute

Setting up the transaction log

Before you create the Replication Agent transaction log, you can specify the object name prefix string that will be used to name all the transaction log tables.

The pdb_xlog_prefix configuration parameter stores the prefix string. Use the ra_config command to change the value of the pdb_xlog_prefix parameter.

When the pdb_xlog_prefix parameter is set to use the prefix string you want for the Replication Agent transaction log tables, you can use the pdb_xlog command to create the transaction log base objects in the primary database.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log tables in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, the Replication Agent instance will not be able to find the transaction log tables that use the old prefix.

Administering the transaction log

The Replication Agent for UDB does not support backing up or restoring transaction logs or truncating transaction logs. All DB2 Universal Database logs are maintained through the database server.

Archiving the transaction log

Before you archive any DB2 Universal Database transaction logs in a primary database, you must make sure that the Replication Agent has finished processing all replicated operations in the log using the following procedure.

❖ **To determine whether operations in the log have been processed by the Replication Agent**

1 With the Replication Agent in *Replicating* state, use the ra_locator command to retrieve the current value of the LTM locator from the primary Replication Server.

```
ra_locator update
```

The current value of the LTM locator from the primary Replication Server contains the DB2 Universal Database log sequence number of the last transaction operation to be replicated successfully.

Characters 4-19 of the origin queue ID are the 16-character (8-byte) operation sequence number. Characters 8-19 (6 bytes) of the operation sequence number are the DB2 log sequence number.

2 Use the DB2 db2flsn command, with the log sequence number identified in the origin queue ID, to identify which log file contains that log sequence number.

You can archive any log files up to the log file name returned by the db2flsn command.

Replication Agent for UDB setup test scripts

Sybase Replication Agent provides a set of test scripts that automate the process of creating a replication test environment that you can use to verify the installation and configuration of the Replication Agent software and the basic function of the other components in your replication system.

The Replication Agent test scripts are located in the *scripts* subdirectory under the Replication Agent base directory. For example: *rax-12_5\scripts*.

The Replication Agent test scripts perform the following tasks:

1 Create a primary table.

2 Create a replicate table that corresponds to the primary table.

3 Create the primary database server connection in the primary Replication Server.

4 Create the replication definition in the primary Replication Server.

5 Test the replication definition.

6 Create the subscription in the replicate Replication Server.

7 Test the subscription.

8 Create the Replication Agent transaction log tables in the primary database.

9 Modify the primary table.

10 Clean up and remove the replication test environment created by the other scripts.

Before you begin

Before running the test scripts, make sure that you have:

- Installed an IBM DB2 Universal Database server
- Installed a database server to act as a replicate data server
- Created the replicate database in the replicate data server
- Installed primary and replicate Replication Servers (PRS and RRS)

Note The PRS and RRS can be the same Replication Server. You must create routes between them if the PRS and RRS are not the same Replication Server.

- Added the replicate database as an RRS-managed database (created a database connection for the replicate database in the RRS)
- Installed the Sybase Replication Agent software, created a Replication Agent for UDB instance, and used the ra_config command to set the following configuration parameters:
  - ra_config ltl_character_case, lower
  - ra_config rs_source_ds, rax
  - ra_config rs_source_db, test

Note These recommended configuration parameter values are for this test only. The values you should use in a production environment (or even a different test environment) may be different.

- Configured all the pds, rs, and rssd connection parameters and tested them successfully with the Replication Agent test_connection command
- Confirmed that all servers are running
- Confirmed that the Replication Agent for UDB instance is in the *Admin* state

Create the primary table

Use your database access tool (such as Command Line Processor) to log in to the DB2 Universal Database server and execute the udb_create_test_primary_table.sql script to create the primary table in the primary database.

Create the replicate table

Use isql or another query processor to log in to the replicate database server and execute the ase_create_test_replicate_table.sql script to create the replicate table in the replicate database.

Create the Replication Server connection for Replication Agent

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_connection.sql script to create the Replication Server connection to the primary database.

Create the replication definition

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_repdef.sql script to create a test replication definition.

Test the replication definition

Use isql or another query processor to log in to the RSSD of the primary Replication Server and execute the rssd_helprep_test_repdef.sql script to test the replication definition.

Create the subscription

There are two scripts provided for this step:

- rs_create_test_sub.sql — designed for use with Replication Server version 11.5 or later

- rs_create_test_sub_for_11.0.x.sql — designed for use with Replication Server version 11.0.x

❖ **To create the test subscription**

1 Edit the appropriate script file (rs_create_test_sub.sql or rs_create_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used to create the connection to the replicate data server and replicate database. These values are initially set to ase.test.

2 Use isql or another query processor to access the replicate Replication Server and execute the appropriate script to create the test subscription.

Note If you are using the rs_create_test_sub_for_11.0.x.sql script, the script must be executed twice because, even though the PRS and RRS are the same Replication Server, there is not enough time between the command executions in the script to allow Replication Server to complete all its tasks before the next script command is executed. For this reason, you may wish to execute the different commands in this script separately.

Note This procedure assumes that no materialization is necessary. See Appendix E, “Materializing Subscriptions to Primary Data” for more information about materialization.

Test the subscription

Use isql or another query processor to access the RSSD of the replicate Replication Server and execute the rssd_helpsub_test_sub.sql script to test the subscription.

Create the Replication Agent transaction log

If you have not already created the Replication Agent transaction log in the primary database, create the transaction log now.

❖ **To create the Replication Agent transaction log**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to create the transaction log.

```
pdb_xlog create
```

See "Creating the Replication Agent transaction log" on page 50 for more information.

Mark a primary table for replication

Mark the test primary table (rax_test) that was created in the primary database in the DB2 Universal Database server by the udb_create_test_primary_table.sql script.

❖ **To mark the test primary table for replication**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to mark the rax_test table for replication.

```
pdb_setreptable rax_test, mark
```

See "Marking objects in the primary database" on page 18 for more information.

Note Make sure that replication is enabled for the rax_test table. See "Enabling and disabling replication for marked tables" on page 61 for more information.

Start replication

If you have not already put the Replication Agent instance in the *Replicating* state, put the Replication Agent instance in the *Replicating* state now.

❖ **To start test replication**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to put the Replication Agent instance in the *Replicating* state.

```
resume
```

See "Starting replication" on page 24 for information.

Note Wait 5-10 seconds after you invoke the resume command before you execute the test transaction script.

Execute the test transaction script in the primary database

Use your DB2 Universal Database access tool (such as Command Line Processor) to log in to the DB2 Universal Database server and execute the udb_primary_test_transactions.sql script to generate transactions in the primary table.

Check results of replication

Use your DB2 Universal Database access tool (such as Command Line Processor) to log in to the DB2 Universal Database server and execute the udb_select_test_primary.sql script to view the changes in the test primary table.

Use isql or another query processor to log in to the replicate database and execute the ase_select_test_replicate.sql script to verify that the transactions generated in the primary database were replicated to the test replicate table in the replicate database.

Clean up the test environment

Use the following procedure to clean up and remove the replication test environment that you created in the previous sections.

❖ **To clean up and remove the replication test environment**

1 Log into the Replication Agent administration port using isql (or another query processor).

2 Use the following command to quiesce the Replication Agent instance.

```
quiesce
```

3 Use the following command to remove the Replication Agent transaction log and unmark the test primary table (rax_test) in the primary database.

```
pdb_xlog remove, force
```

4 Log into the replicate Replication Server and execute the following scripts:

   - rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql (to drop the test subscription)

   Edit the appropriate script file (rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used in the rs_create_test_sub.sql script. These values are initially set to ase.test.

   - rs_drop_test_repdef.sql (to drop the test replication definition)
   - rs_drop_test_connection.sql (to drop the test connection to the primary database)

5 Log into the replicate data server and execute the ase_drop_test_replicate_table.sql script to drop the test replicate table.

6 Use your DB2 Universal Database access tool (such as Command Line Processor) to log in to the DB2 Universal Database server and execute the udb_drop_test_primary_table.sql script to drop the test primary table.

APPENDIX B

Administering the Replication Agent for Informix

The term “Replication Agent for Informix” refers to an instance of the Sybase Replication Agent version 12.5 software configured for a primary database that resides on an Informix Dynamic Server database server. This appendix describes the characteristics of the Sybase Replication Agent that are unique to the Replication Agent for Informix implementation.

Note For information on the basic functionality of Sybase Replication Agent version 12.5, refer to the other parts of this *Sybase Replication Agent Administration Guide*.

This appendix includes the following sections:

- Informix-specific issues
- Replication Agent for Informix transaction log
- Replication Agent for Informix setup test scripts

Informix-specific issues

This section describes general issues and considerations that are specific to using Sybase Replication Agent version 12.5 with the Informix Dynamic Server database server.

The following topics are included in this section:

- Replication Agent connectivity
- Transaction isolation
- Informix trigger limitations
- Maximum number of columns in a table
- Transaction commit order

- Change database command limitation
- Character case of database object names
- Stored procedure replication
- Format of origin queue ID
- Datatype compatibility

Replication Agent connectivity

Connectivity between the Replication Agent for Informix and the Informix data server is by way of the Informix JDBC driver.

The Informix JDBC driver must be installed on the Replication Agent host machine and the full path name of this driver must be included in the user's *CLASSPATH* environment variable before starting a Replication Agent instance.

For more information on configuring the Informix JDBC driver, see the *Sybase Replication Agent Installation Guide*.

Transaction isolation

The default Informix database transaction isolation must be Committed Read. It cannot be Uncommitted Read or Transactions not supported.

This is ANSI standard behavior that guarantees that *user2* will see the data only after *user1* has committed the changes. So if a change is in progress, *user2* will be locked out until the change is either committed or rolled back. This is only possible if the database is logged.

Informix trigger limitations

There are three types of trigger limitations in an Informix primary database:

- Selective update triggers are not allowed.
- Disabled triggers are not allowed.
- Reentrant triggers should be avoided.

Selective triggers

No table marked for replication can have a selective update trigger on it. (A selective update trigger is an update trigger that references specific columns.)

In the Informix Dynamic Server, a table may have multiple update triggers, each of which specifies a subset of the columns. However, each column may be referenced by only one update trigger.

Replication Agent requires an update trigger for all columns (an entire row), so if a selective update trigger already exists on the table, Replication Agent cannot create its update trigger, and therefore it cannot replicate transactions against that table.

Note Dynamic Server 2000 allows users to create select triggers. If the select trigger does not modify data in the triggering table, then the selective select trigger is allowed on marked tables.

Disabled triggers

No table marked for replication can have a disabled insert, update, or delete trigger on it.

Even though Replication Agent can modify a disabled trigger, as long as the trigger is disabled, it does not execute when the corresponding operation occurs, and therefore, replication of that operation does not take place.

Reentrant triggers

Dynamic Server 2000 allows users to create reentrant triggers. A reentrant trigger is one where the triggering event and triggered action both act on the same table.

Note Reentrant triggers may cause problems with replication. Therefore, Sybase recommends you avoid using reentrant triggers on tables marked for replication.

Maximum number of columns in a table

A table marked for replication can have no more than four columns less than the maximum number allowed by the Informix Dynamic Server instance.

For example, if the database instance allows up to 232 columns in a table, any table marked for replication can have up to 228 columns. Replication Agent adds four additional columns to a transaction log shadow table, which must not exceed the maximum number of columns allowed by the database instance.

Transaction commit order

The Replication Agent transaction log is implemented using database objects (tables, stored procedures, and triggers) in the primary Informix database. Because of this and the way the Informix database handles transactions internally, there is some risk that transactions may not be applied to a replicate table in the same order they were applied to the primary table.

For example, if transaction *B* commits during the time lapse between the last data change operation and the commit for transaction *A*, Sybase Replication Agent presents the transactions to the primary Replication Server with the order *AB*, when the actual commit order was *BA*.

To avoid this situation, you can do one of the following:

- Ensure that commits are performed immediately after the last data-changing operation in a transaction.
- Perform a dummy data-altering operation to a marked table just before committing a transaction.

Change database command limitation

Sybase Replication Agent provides a command that allows you to specify the current database for executing SQL commands (pdb_set_sql_database) and a command that allows you to view a list of the available databases on the data server Replication Agent is connected to (pdb_get_databases).

Because of limitations in the functionality of the Informix JDBC driver, both of these Replication Agent commands behave as if the Informix database server has only one database:

- The pdb_set_sql_database command returns success, but the current database does not change.

- The pdb_get_databases command returns only the name of the current database.

Character case of database object names

Database object names must be delivered to the primary Replication Server in the same format as they are specified in replication definitions. For example, if a replication definition specifies a table name in all uppercase, then that table name must appear in all uppercase when it is sent to the primary Replication Server by the Replication Agent.

Note Replication will fail if a database object name is delivered to the primary Replication Server in a format different from that specified in the replication definition.

Sybase Replication Agent version 12.5 gives you some control over the way it treats the character case of database object names when it sends LTL to the primary Replication Server. You have three options:

- as-is — database object names are passed to Replication Server in the same format as they are actually stored in the primary database server
- lower — database object names are passed to Replication Server in *all lowercase*, regardless of the way they are actually stored in the primary database server
- upper — database object names are passed to Replication Server in *all uppercase*, regardless of the way they are actually stored in the primary database server

You specify the character case option you want by setting the value of the ltl_character_case configuration parameter. The parameter values are asis, lower, and upper. The default value of the ltl_character_case parameter is asis.

In the case of the Informix database server, database object names are stored in all lowercase. Therefore, the asis and lower options have the same affect on database object names sent from Replication Agent for Informix to the primary Replication Server.

Stored procedure replication

There are two issues related to stored procedure replication from a Dynamic Server 2000 primary database:

- Replicating user-defined routines (UDRs) from a Dynamic Server 2000 primary database
- UDR syntax

Replicating UDRs

The following limitations apply to replicating user-defined routines (UDRs) from a primary database in Dynamic Server 2000:

- UDRs written in an external language, such as C or Java, cannot be replicated because the Replication Agent cannot access external UDR source. To be replicated, a UDR must be written in SPL (stored procedure language).
- When searching the primary database for the UDR named in a pdb_setrepproc *proc*, mark command, the Replication Agent searches for a UDR with the specific name of *proc* or with the actual name of *proc* (and if *owner* is specified, *owner.proc*), using it in the where clause. If no UDR is found, or if more than one UDR is found, the specified UDR cannot be marked for replication as a stored procedure.

  If the specific-name search succeeds and no replicate name was supplied in the pdb_setrepproc command, the replicate name will be the actual UDR name.

UDR syntax

To be able to mark a UDR for replication, the Replication Agent must be able to distinguish between the routine header and the routine body.

In versions of the Informix Dynamic Server prior to Dynamic Server 2000 (version 9.2), stored procedure syntax required a semi-colon at the end of the returning clause, or the header ended with the close parenthesis following the parameter list.

In Dynamic Server 2000, the semi-colon at the end of the header is optional. Without the semi-colon, or without a begin label, the Replication Agent cannot distinguish the routine header from the routine body.

If the routine has any header syntax elements following the parameter list, such as the specific name, the semi-colon must be used at the end of all the header options, or a begin label must be used to designate the beginning of the statement block. Otherwise, Replication Agent cannot mark the UDR for replication.

Format of origin queue ID

Each record in the transaction log is identified by an origin queue ID that consists of 64 hexadecimal characters (32 bytes). The format of the origin queue ID is determined by the Replication Agent instance, and it varies according to the primary database type.

Table B-1 illustrates the format of the origin queue ID for the Replication Agent for Informix with Informix Dynamic Server version 7.x.

Table B-1: Replication Agent for Informix origin queue ID for Dynamic Server version 7.x

Character	Bytes	Description
0-3	2	database generation ID
4-19	8	transaction max sequence
20-27	4	operationID HI
28-35	4	operationID LO
36-39	2	operation type (begin = 0, data/LOB = 1, commit/rollback = 7FFF)
40-55	8	transaction ID
56-63	4	LOB sequence ID

Table B-2 illustrates the format of the origin queue ID for the Replication Agent for Informix with Informix Dynamic Server 2000 (version 9.2).

Table B-2: Replication Agent for Informix origin queue ID for Dynamic Server 2000 (version 9.2)

Character	Bytes	Description
0-3	2	database generation ID
4-19	8	transaction max sequence
20-35	8	operationID
36-39	2	operation type (begin = 0, data/LOB = 1, commit/rollback = 7FFF)
40-55	8	transaction ID
56-63	4	LOB sequence ID

Datatype compatibility

Replication Agent for Informix processes transactions and stored procedure invocations and passes data to the primary Replication Server.

The primary Replication Server uses the datatype formats specified in the replication definition to receive the data from Replication Agent for Informix.

The following table describes the default conversion of Informix datatypes to Sybase datatypes.

Table B-3: Informix to Sybase default datatype mapping

Informix datatype	Informix length/range	Sybase datatype	Sybase length/range	Notes
byte	2^{31} bytes, binary data	image	2GB	
blob	4 terabytes ($4*2^{40}$), binary data	image	2GB	Values over 2GB are truncated.
boolean	1 byte	char(1)	1 byte	Values are replicated as literal characters t or f.
char(n)	32,767 bytes	char, varchar	32K	
clob	4 terabytes ($4*2^{40}$), text data	text	2GB	Values over 2GB are truncated.
date	number of days since 12/31/1899	datetime	01/01/1753 to 12/31/9999	No special effort is required to format the value for either Replication Server or Adaptive Server. If the Informix value falls outside the Replication Server supported range, Replication Server will not accept it. Options for handling this situation include: • Using char(10) as the datatype in the replication definition. • Modifying the triggers and stored procedures that support the shadow table to normalize values outside Replication Server limits.

Informix datatype	Informix length/range	Sybase datatype	Sybase length/range	Notes
datetime *largest_qualifier* to *smallest_qualifier*	YEAR: 1 - 9999 MONTH: 1 - 12 DAY: 1 - 31 HOUR: 0 - 23 MINUTE: 0 - 59 SECOND: 0 - 59 FRACTION: 0 - 99999	char(n) or datetime	If datetime is used, 01/01/1753 to 12/31/9999	If char is used, the value of *n* depends on the date elements defined for the column. For example, datetime year to fraction(5) requires *n* to be 25, datetime month to day requires *n* to be 5. If datetime is used, set the value of the pdb_convert_datetime parameter to true before marking the table, then deal with the range issue as described for the Informix date datatype.
decimal(p) or dec(p)	10^{-130} to 10^{124}, up to 32 significant digits	decimal or float	When decimal is used, 10^{-38} to 10^{38}, 38 significant digits. float precision and range corresponds to a C double datatype, approximately 16 significant digits.	Using the decimal datatype allows precision at the expense of range. Using the float datatype allows range (to the extent that the Replication Server platform supports range) at the expense of precision. However, extremely small values get truncated at log time to 16 digits to the right of the decimal. For example, with the primary column datatype of decimal(32), the number 1.2345678901234567890123456789012e-10 is logged as 0.0000000001234567. Extremely large values retain precision.
decimal(p,s) or dec(p,s) or numeric(p,s)	Range without error is 0 to 10^{p-s} - 10^{-s}, up to 32 significant digits	decimal	10^{-38} to 10^{38}, 38 significant digits.	

Informix datatype	Informix length/range	Sybase datatype	Sybase length/range	Notes
float(n) or double precision	Corresponds to a C double datatype on the given system, approximately 16 significant digits.	float	Corresponds to a C double datatype on the given system, approximately 16 significant digits.	Errors can be introduced if the Informix platform and Replication Server platform handle the C double datatype differently. Extremely large values may lose significant digits at log time. For example, with primary column datatype of float, the number 1.234567890123456e125 is logged as 1.234568e+125. However, extremely small values retain all digits.)
integer or int	-2,147,483,647 to 2,147,483,647.	int	-2,147,483,648 to 2,147,483,647.	
int8	-9,223,372,036, 854,775,807 to 9,223,372,036, 854,775,807	decimal	10^{-38} to 10^{38}, 38 significant digits.	
interval *largest_qualifier*(n) to *smallest_qualifier*(n)	1 <= *n* <= 9 for *largest_qualifier*. If the type of *smallest_qualifier* is fraction, 1 <= *n* <= 5. (*n* on *smallest_qualifier* only valid if *smallest_qualifier* is a fraction.)	char(n)	32K	The value of the char *n* depends on the interval elements defined for the column. For example, interval year(5) to month requires *n* to be 9, interval day(9) to fraction(5) requires *n* to be 25. By modifying the script that builds the shadow table and associated primary table triggers and stored procedures, you can break the interval value into separate integer values. Or you can use a function string to do this breakdown for each replicate.
lvarchar	4 GB	text	2GB	Values over 2GB are truncated.

Informix datatype	Informix length/range	Sybase datatype	Sybase length/range	Notes
money(p,s)	Range without error is 0 to 10^{p-s} - 10^{-s}, up to 32 significant digits	money or small-money	The money range is -922,337,203, 685,477.5808 to 922,337,203, 685,477.5807. The smallmoney range is -214,748.3648 to 214,748.3647.	Both money and smallmoney datatypes are accurate to one ten-thousandth of a monetary unit. This results in truncation if the primary column allows accuracy beyond that level.
nchar(n)	up to 32,767 bytes of multi-byte character data	unichar or char	32K	To use the unichar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
nvarchar(m,r)	up to 255 bytes of multi-byte character data	univarchar or varchar	32K	To use the univarchar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
real	See smallfloat.			

serial	-2,147,483,647 to 2,147,483,647.	identity or integer	-2,147,483,648 to 2,147,483,647.	If identity is used, the following command is applied to the replicated table before an insert command:
				`set identity_insert` *`table_name`* `on`
				The following command is applied to the replicated table after an insert command:
				`set identity_insert` *`table_name`* `off`
				Note: Columns with the identity datatype are never updated by the update command.

Informix datatype	Informix length/range	Sybase datatype	Sybase length/range	Notes

serial8	-9,223,372,036, 854,775,807 to 9,223,372,036, 854,775,807	identity or numeric	1 to 10^{38} - 1	If identity is used, the following command is applied to the replicated table before an insert command:
				`set identity_insert` *table_name* `on`
				The following command is applied to the replicated table after an insert command:
				`set identity_insert` *table_name* `off`
				Note: Columns with the identity datatype are never updated by the update command.

smallfloat	Corresponds to a C float datatype, approximately 8 significant digits.	real	Corresponds to a C float datatype, approximately 6 significant digits.	Errors may be introduced if the Informix platform and Replication Server platform handle the C float datatype differently.
smallint	-32,767 to 32,767	smallint	-32,768 to 32,767	
text	2 GB, text data.	text	2GB	
varchar(m [,r])	255 bytes	char or varchar	32K	

For each datatype in Table B-3, lengths in the second column are described as:

- Character and graphic datatypes – maximum number of bytes
- Numeric and temporal datatypes – range from smallest to largest values

Informix datatype restrictions

Replication Server and Replication Agent impose the following constraints on Informix datatypes:

- Informix blob, clob, and lvarchar datatypes that contain more than 2GB will be truncated to 2GB.
- If you use a version of Replication Server prior to version 12.5:
  - char and varchar dataypes that contain more than 255 bytes will be truncated to 255 bytes.

- nchar and nvarchar multi-byte character dataypes will be replicated as char or varchar single-byte datatypes.

Replication Agent for Informix transaction log

Sybase Replication Agent uses its own proprietary transaction log to capture and record transactions in the primary database for replication. The Replication Agent transaction log consists of a set of shadow tables, stored procedures, and triggers that are created in the primary database.

The Replication Agent transaction log is created by invoking the pdb_xlog command with the create keyword. When you invoke this command, Replication Agent generates a SQL script that is run in the primary database. This script (stored in the *create.sql* file in the *rax-12_5\inst_name\scripts\xlog* directory) creates the Replication Agent transaction log components referred to as the *transaction log base objects*. The transaction log base objects must be created before any objects can be marked for replication in the primary database.

Note This section describes the schema and details of the Replication Agent transaction log for an Informix database. For more general information about the Replication Agent transaction log, see Chapter 3, "Administering Sybase Replication Agent," in this book.

Transaction log components

There are three types of Informix database objects used for transaction log components:

- tables
- stored procedures
- triggers

There are several tables used by the Replication Agent transaction log, both as base objects and as shadow tables for marked primary tables in the primary database. All table components of the transaction log contain at least one index, and some contain more than one index.

Stored procedures are used by the Replication Agent transaction log to capture transactions for replication and to convert the datetime data type.

Triggers are used to capture the insert, update, and delete operations in marked primary tables.

Transaction log object names

There are two variables in the transaction log component database object names shown in this appendix:

- *prefix* — represents the one- to three-character string value of the pdb_xlog_prefix parameter (the default is ra_).
- *xxx* — represents an alphanumeric counter, a string of characters that is (or may be) added to a database object name to make that name unique in the database.

The value of the pdb_xlog_prefix parameter is the prefix string used in all Replication Agent transaction log component names.

If this value conflicts with the names of existing database objects in your primary database, you can change the value of the pdb_xlog_prefix parameter by using the ra_config command.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log objects in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, Replication Agent will not be able to find the transaction log objects that use the old prefix.

You can use the pdb_xlog command to view the names of Replication Agent transaction log components in the primary database.

Transaction log base tables

Table B-4 lists the tables that are considered Replication Agent transaction log base objects. No permissions are granted on these tables when they are created.

Table B-4: Replication Agent Transaction Log Base Tables

Table	Database Name
transaction log system table	*prefix*xlog_system_
marked objects table	*prefix*marked_objs_*xxx*
transaction log table	*prefix*tran_log_*xxx*
LOB columns table	*prefix*blob_columns_*xxx*

Table	Database Name
exception holding table	*prefix*exception_*xxx*
Log Reader work table	*prefix*lr_reptran_*xxx*
Log Admin work table	*prefix*rawork_*xxx*
proc active table	*prefix*procactive_*xxx*

Transaction log stored procedures

Table B-5 lists the stored procedures that are considered Replication Agent transaction log base objects. Execute permission is granted to public when these stored procedures are created (except as noted).

Note The transaction log stored procedures listed in Table B-5 have no effect when executed outside the context of replication.

Table B-5: Replication Agent Transaction Log Stored Procedures

Procedure	Database Name
datetime conversion	*prefix*convert_dt_*xxx*
transaction information capture	*prefix*get_tx_info_*xxx*
transaction log row	*prefix*txlog_row_*xxx*
Log Reader scanning	*prefix*bld_reptran_*xxx* (no permission granted)
rollover handling	*prefix*rollover_*xxx* (no permission granted)

Marker procedures and shadow tables

Table B-6 lists the marker procedures and marker shadow tables that are considered Replication Agent transaction log base objects. Execute permission is granted to public when the marker procedures are created.

Table B-6: Replication Agent Marker Procedures and Shadow Tables

Procedure/Table	Database Name
transaction log-marker procedure	rs_marker*xxx*
transaction log-marker shadow table	*prefix*markersh_*xxx*
dump marker procedure	rs_dump*xxx*
dump marker shadow table	*prefix*dumpsh_*xxx*

Transaction log objects for each marked table

Table B-7 lists the Replication Agent transaction log objects that are created for each primary table that is marked for replication. These objects are created only when a table is marked for replication. These objects are not considered transaction log base objects.

No permission is granted on these procedures.

Table B-7: Replication Agent Transaction Log Objects for Each Marked Table

Object	Database Name
shadow table	*prefix*sh_*xxx*
shadow row procedure	*prefix*srp_*xxx*
LOB shadow table	*prefix*bsh_*xxx*
LOB shadow row procedure	*prefix*bsrp_*xxx*
insert trigger	*prefix*instrg_*xxx* (or existing-trigger name)
update trigger	*prefix*updtrg_*xxx* (or existing-trigger name)
delete trigger	*prefix*deltrg_*xxx* (or existing-trigger name)

Note The marked objects table records the mapping between marked objects and their corresponding shadow tables and shadow row procedures. The pdb_setrepproc and pdb_setreptable commands return this information.

Reserved names

There are two type of reserved names in the Replication Agent transaction log:

- Column names in Replication Agent shadow tables
- Global variables in Replication Agent stored procedures

Shadow table column names

When a primary table is marked for replication, Replication Agent creates a shadow table to record the replicated operations. The shadow table has all the columns of the primary table, plus four columns that the Replication Agent creates for its use:

- ra_tran_id_
- ra_opid_hi
- ra_opid_lo
- ra_img_type_

Note These column names are fixed, not generated, therefore the "ra_" portion of the column name does not change when the value of the pdb_xlog_prefix parameter changes.

If a primary table has a column with the same name as one of the Replication Agent shadow table columns, the table marking procedure fails. Replication Agent flags the offending column in the table-marking script and the pdb_setreptable command returns an error.

In this event, you must change the name of the conflicting column in the primary table. After you change the name of the conflicting column, you can modify the table-marking script and run it manually to mark the primary table.

Note If you want to use the primary table's original column name in the replicate table, you can create a function string in the replicate Replication Server to map the new column name in the primary table to the original column name in the replicate table.

Global variables

The following global variables are used in Replication Agent transaction log stored procedures:

- ra_txid_
- ra_opid_hi_
- ra_opid_lo_

- ra_tstamp_

Note These variable names are fixed, not generated, therefore the "ra_" portion of the variable name does not change when the value of the pdb_xlog_prefix parameter changes.

You cannot use these names as column names in a primary table or as global variables in a primary stored procedure in the primary database.

Marked table triggers

If a trigger exists on a primary table when that table is marked for replication, the existing trigger is modified to add a *for-each-row* section (if the trigger does not already contain a *for-each-row* section) to support Sybase replication.

The Sybase replication action in a trigger begins with a comment that serves as a marker so that, when the table is unmarked, the Sybase replication action can be removed from the trigger. If the resulting *for-each-row* section is empty, it will also be removed. If removing the *for-each-row* section leaves the trigger empty, the trigger will be dropped.

Note Do not remove or alter the Sybase replication comments in triggers. If you need to modify a trigger that contains the Sybase replication action, you must keep the Sybase replication action at the end of the trigger (the last action in the trigger).

A "selective update" trigger is an update trigger that acts based on operations only in specified columns.

If a "selective update" trigger exists on a table, Replication Agent cannot modify the trigger and changes to that table cannot be replicated. For more information on the "selective trigger" limitation, see "Selective triggers" on page 227.

Getting actual names of the transaction log objects

Because Sybase Replication Agent generates the names of its transaction log objects using its own algorithms, there is not any obvious correlation between the name of a primary object and the names of the transaction log objects (such as shadow tables) that record replicated operations for the primary object. If you want to find out the names of transaction log objects associated with a primary object, use the following procedure.

❖ **To find out the names of transaction log objects associated with a primary object**

1 At the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

```
pdb_xlog
```

The pdb_xlog command returns a list of the transaction log base objects.

2 At the Replication Agent administration port, invoke the pdb_setrepproc command with no keywords:

```
pdb_setrepproc
```

The pdb_setrepproc command returns a list of all stored procedures marked for replication, including all the Replication Agent transaction log objects associated with each marked procedure.

3 At the Replication Agent administration port, invoke the pdb_setreptable command with no keywords:

```
pdb_setreptable
```

The pdb_setreptable command returns a list of all tables marked for replication, including all the Replication Agent transaction log objects associated with each marked table.

Marked objects table

One of the Replication Agent transaction log base objects is the **marked objects table**. The marked objects table contains an entry for each marked object in the primary database (both tables and stored procedures). Each marked object entry contains the following information:

- the name of the marked primary object
- the primary object's replicated name (if any)
- the type of the primary object
- the "replication enabled" flag for the primary object
- the name of the shadow table for the primary object
- the name of the operation-image procedure
- the name of the LOB shadow table (if the primary object is a LOB column)
- the name of the LOB image procedure (if the primary object is a LOB column)

- the owner of the primary object
- the “send owner” flag

Setting up the transaction log

Before you create the Replication Agent transaction log base objects, you can specify the object name prefix string that will be used to name all the transaction log objects.

The pdb_xlog_prefix configuration parameter stores the prefix string. Use the ra_config command to change the value of the pdb_xlog_prefix parameter.

When the pdb_xlog_prefix parameter is set to use the prefix string you want for the Replication Agent transaction log objects, you can use the pdb_xlog command to create the transaction log base objects in the primary database.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, Replication Agent will not be able to find the transaction log objects that use the old prefix.

Administering the transaction log

The only transaction log administration required is backing up the transaction log and truncation.

Backing up and restoring the transaction log

Sybase Replication Agent does not support backing up and restoring the transaction log automatically.

Sybase recommends that you use the database backup utilities provided with your Informix Dynamic Server software to periodically back up the transaction log.

Note Sybase Replication Agent does not support replaying transactions from a restored log.

Truncating the transaction log

Sybase Replication Agent provides features for both automatic and manual log truncation.

Replication Agent provides two options for automatic transaction log truncation:

- Periodic truncation, based on a time interval you specify.
- Automatic truncation whenever Replication Agent receives a new LTM Locator value from the primary Replication Server.

You also have the option to switch off automatic log truncation. By default, automatic log truncation is switched off.

You can specify the automatic truncation option you want (including none) by using the ra_config command to set the value of the truncation_type configuration parameter.

If you want to truncate the transaction log automatically based on a time interval, use the ra_config command to set the value of the truncation_interval configuration parameter.

You can truncate the Replication Agent transaction log manually, at any time, by invoking the pdb_truncate_xlog command at the Replication Agent administration port.

If you want to truncate the transaction log at a specific time, you can use a scheduler utility to execute the pdb_truncate_xlog command automatically.

Replication Agent for Informix setup test scripts

Sybase Replication Agent provides a set of test scripts that automate the process of creating a replication test environment that you can use to verify the installation and configuration of the Replication Agent software and the basic function of the other components in your replication system.

The Replication Agent test scripts are located in the *scripts* subdirectory under the Replication Agent base directory. For example: *rax-12_5\scripts*.

The Replication Agent test scripts perform the following tasks:

1 Create a primary table.

2 Create a replicate table that corresponds to the primary table.

3 Create the primary database server connection in the primary Replication Server.

4 Create the replication definition in the primary Replication Server.

5 Test the replication definition.

6 Create the subscription in the replicate Replication Server.

7 Test the subscription.

8 Create the Replication Agent transaction log in the primary database.

9 Modify the primary table.

10 Clean up and remove the replication test environment created by the other scripts.

Before you begin

Before running the test scripts, make sure that you have:

- Installed an Informix database server
- Installed a database server to act as a replicate data server
- Created the replicate database in the replicate data server
- Installed primary and replicate Replication Servers (PRS and RRS)

  Note The PRS and RRS can be the same Replication Server. You must create routes between them if the PRS and RRS are not the same Replication Server.

- Added the replicate database as an RRS-managed database
- Installed the Replication Agent software, created a Replication Agent instance, and used the ra_config command to set the following configuration parameters:
  - ra_config ltl_character_case, lower
  - ra_config rs_source_ds, rax
  - ra_config rs_source_db, test

  Note These recommended configuration parameter values are for this test only. The values you should use in a production environment (or even a different test environment) may be different.

- Configured all the pds, rs, and rssd connection parameters and tested them successfully with the Replication Agent test_connection command
- Confirmed that all servers are running
- Confirmed that the Replication Agent instance is in the *Admin* state

Create the primary table

Use your Informix database access tool (such as dbaccess) to log in to the Informix data server and execute the informix_create_test_primary_table.sql script to create the primary table in the primary database.

Create the replicate table

Use isql or another query processor to log in to the replicate database server and execute the ase_create_test_replicate_table.sql script to create the replicate table in the replicate database.

Create the Replication Server connection for Replication Agent

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_connection.sql script to create the Replication Server connection to the primary database.

Create the replication definition

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_repdef.sql script to create a test replication definition.

Test the replication definition

Use isql or another query processor to log in to the RSSD of the primary Replication Server and execute the rssd_helprep_test_repdef.sql script to test the replication definition.

Create the subscription

There are two scripts provided for this step:

- rs_create_test_sub.sql — designed for use with Replication Server version 11.5 or later
- rs_create_test_sub_for_11.0.x.sql — designed for use with Replication Server version 11.0.x

❖ **To create the test subscription**

1 Edit the appropriate script file (rs_create_test_sub.sql or rs_create_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that

you used to create the connection to the replicate data server and replicate database. These values are initially set to ase.test.

2 Use isql or another query processor to access the replicate Replication Server and execute the appropriate script to create the test subscription.

Note If you are using the rs_create_test_sub_for_11.0.x.sql script, the script must be executed twice because, even though the PRS and RRS are the same Replication Server, there is not enough time between the command executions in the script to allow Replication Server to complete all its tasks before the next script command is executed. For this reason, you may wish to execute the different commands in this script separately.

Note This procedure assumes that no materialization is necessary. See Appendix E, "Materializing Subscriptions to Primary Data," for more information about materialization.

Test the subscription

Use isql or another query processor to access the RSSD of the replicate Replication Server and execute the rssd_helpsub_test_sub.sql script to test the subscription.

Create the Replication Agent transaction log

If you have not already created the Replication Agent transaction log in the primary database, create the transaction log now.

❖ **To create the Replication Agent transaction log**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to create the transaction log.

```
pdb_xlog create
```

See "Creating the Replication Agent transaction log" on page 16 for more information.

Mark a primary table for replication

Mark the test primary table (rax_test) that was created in the Informix database by the informix_create_test_primary_table.sql script.

❖ **To mark the test primary table for replication**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to mark the rax_test table for replication.

```
pdb_setreptable rax_test, mark
```

See "Marking objects in the primary database" on page 18 for more information.

Note Make sure that replication is enabled for the rax_test table. See "Enabling and disabling replication for marked tables" on page 61 for more information.

Start replication

If you have not already put the Replication Agent instance in the *Replicating* state, put the Replication Agent instance in the *Replicating* state now.

❖ **To start test replication**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to put the Replication Agent instance in the *Replicating* state.

```
resume
```

See "Starting replication" on page 24 for information.

Execute the test transaction script in Informix

Use your Informix database access tool (such as dbaccess) to log in to the Informix data server and execute the informix_primary_test_transactions.sql script to generate transactions in the primary table in the primary database.

Check results of replication

Use your Informix database access tool (such as dbaccess) to log in to the Informix data server and execute the informix_select_test_primary.sql script to view the changes in the test primary table in the primary database.

Use isql or another query processor to log in to the replicate database and execute the ase_select_test_replicate.sql script to verify that the transactions generated in the primary database were replicated to the test replicate table in the replicate database.

Clean up the test environment

Use the following procedure to clean up and remove the replication test environment that you created in the previous sections.

❖ **To clean up and remove the replication test environment**

1 Log into the Replication Agent administration port using isql (or another query processor).

2 Use the following command to quiesce the Replication Agent instance.

```
quiesce
```

3 Use the following command to remove the Replication Agent transaction log and unmark the test primary table (rax_test) in the Informix database.

```
pdb_xlog remove, force
```

4 Log into the replicate Replication Server and execute the following scripts:

- rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql (to drop the test subscription)

  Edit the appropriate script file (rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used in the rs_create_test_sub.sql script. These values are initially set to ase.test.

- rs_drop_test_repdef.sql (to drop the test replication definition)
- rs_drop_test_connection.sql (to drop the test connection to the primary database)

5 Log into the replicate data server and execute the ase_drop_test_replicate_table.sql script to drop the test replicate table.

6 Use your Informix database access tool (such as dbaccess) to log in to the Informix data server and execute the informix_drop_test_primary_table.sql script to drop the test primary table.

APPENDIX C

Administering the Replication Agent for Microsoft SQL Server

The term “Replication Agent for Microsoft SQL Server” refers to an instance of the Sybase Replication Agent version 12.5 software installed and configured for a primary database that resides on a Microsoft SQL Server database server. This appendix describes the characteristics of the Sybase Replication Agent that are unique to the Replication Agent for Microsoft SQL Server implementation.

Note For information on the basic functionality of Sybase Replication Agent version 12.5, refer to the other parts of this *Sybase Replication Agent Administration Guide*.

This appendix includes the following sections:

- Microsoft SQL Server-specific issues
- Replication Agent for Microsoft SQL Server transaction log
- Replication Agent for Microsoft SQL Server setup test scripts

Microsoft SQL Server-specific issues

This section describes general issues and considerations that are specific to using Sybase Replication Agent version 12.5 with the Microsoft SQL Server database server.

Note In this appendix, the term “Windows” refers to both Windows NT and Windows 2000 operating systems.

The following topics are included in this section:

- Replication Agent communications
- Replication Agent permissions
- Maximum number of columns in a table
- @@IDENTITY system variable
- Length of owner names
- Microsoft isql tool
- Character case of database object names
- Format of origin queue ID
- Datatype compatibility

Replication Agent communications

Replication Agent for Microsoft SQL Server uses the Java Database Connectivity (JDBC) protocol for communications with all replication system components.

Replication Agent for Microsoft SQL Server connects to Microsoft SQL Server using the JDBC driver provided with the Sybase Replication Agent software.

Replication Agent for Microsoft SQL Server uses Sybase jConnect for JDBC to communicate with the primary Replication Server and its RSSD.

While replicating transactions and function invocations, Replication Agent maintains connections with both the primary database and the primary Replication Server. In addition, Replication Agent occasionally connects to the RSSD of the primary Replication Server to retrieve replication definition data.

For more information about Replication Agent communications, see Chapter 1, “Introduction to Sybase Replication Agent,” in this book.

Replication Agent permissions

Replication Agent for Microsoft SQL Server must create database objects (tables, triggers, and procedures) in the primary database for its transaction log.

The user ID that the Replication Agent instance uses to log in to the Microsoft SQL Server must have access to the primary database with the following permissions granted:

- Create table
- Create trigger
- Create procedure

Note Create trigger permission cannot be transferred from the table owner in Microsoft SQL Server 6.5. If you use Microsoft SQL Server 6.5 with Replication Agent for Microsoft SQL Server, you may have to create transaction log triggers by manually running the trigger-creation scripts using the table owner's user ID for each replicated table.

Maximum number of columns in a table

A table marked for replication can have no more than three columns less than the maximum number allowed by the Microsoft SQL Server database.

For example, if the database allows up to 232 columns in a table, any table marked for replication can have up to 229 columns. Replication Agent adds three additional columns to a transaction log shadow table, which must not exceed the maximum number of columns allowed by the database.

@@IDENTITY system variable

Microsoft SQL Server has a system variable named *@@IDENTITY*, which stores the most recent value of the identity column after an insert operation. When a Replication Agent trigger executes to capture an insert operation in a marked table, that trigger performs an insert operation in one of the transaction log shadow tables. As a result of the trigger execution, the value of the *@@IDENTITY* variable after an insert operation in a marked table is actually the value after the trigger's insert in the shadow table. Since the Replication Agent shadow table does not have an identity column, the value of the *@@IDENTITY* variable is set to NULL.

If you have a SQL Server application that uses the *@@IDENTITY* variable, use the following procedure to work around this problem:

❖ **To work around the @@*IDENTITY* variable problem:**

1 Create a table in the primary database that has a single column defined as integer identity.

This column will be used to store the original value of the *@@IDENTITY* variable.

2 Modify the Replication Agent insert trigger (*prefix*instrg_*xxx*) as follows:

- Create a local variable in the Replication Agent insert trigger (*prefix*instrg_*xxx*) to hold the value of the *@@IDENTITY* variable.
- Immediately save the value of the *@@IDENTITY* variable in the trigger's local variable before performing any replication operations.
- After all replication operations are complete, use the value saved in the trigger's local variable to insert a new record in the identity column of the new table created for this purpose. (You have to temporarily turn on the Identity_Insert option to insert an explicit value in the identity column, then turn it off after the insert.)

Note To execute a trigger that contains the Identity_Insert command, the Replication Agent user ID must have sa or table owner permissions.

After the original value of the *@@IDENTITY* variable is inserted in the identity column in the new table, the value of the *@@IDENTITY* variable is restored.

Length of owner names

The maximum length of owner names is increased to 128 characters in Microsoft SQL Server 2000. Replication Server 12.5 supports user (table or procedure owner) names up to 30 bytes.

Microsoft isql tool

The database access tool provided with Microsoft SQL Server is Microsoft isql. You must use Microsoft isql (or a compatible tool) to access the Microsoft SQL Server database to execute some of the test scripts documented in this Appendix.

Although the name of the Microsoft isql tool is the same as the Sybase tool called isql, the Sybase and Microsoft tools are not compatible. For example, you cannot use the Sybase isql tool to access the Microsoft SQL Server data server and you cannot use the Microsoft isql tool to access the Replication Agent administration port.

If you have both Sybase and Microsoft isql tools loaded on the same computer, you may need to change an environment variable (possibly the *path* variable) to avoid problems when you invoke one of the isql tools.

Character case of database object names

Database object names must be delivered to the primary Replication Server in the same format as they are specified in replication definitions. For example, if a replication definition specifies a table name in all uppercase, then that table name must appear in all uppercase when it is sent to the primary Replication Server by the Replication Agent.

Note Replication will fail if a database object name is delivered to the primary Replication Server in a format different from that specified in the replication definition.

Sybase Replication Agent version 12.5 gives you some control over the way it treats the character case of database object names when it sends LTL to the primary Replication Server. You have three options:

- as-is — database object names are passed to Replication Server in the same format as they are actually stored in the primary database server
- lower — database object names are passed to Replication Server in *all lowercase*, regardless of the way they are actually stored in the primary database server
- upper — database object names are passed to Replication Server in *all uppercase*, regardless of the way they are actually stored in the primary database server

You specify the character case option you want by setting the value of the ltl_character_case configuration parameter. The parameter values are asis, lower, and upper. The default value of the ltl_character_case parameter is asis.

In the case of Microsoft SQL Server, database object names are stored in the same case as entered (upper and/or lower). Therefore, you must use the asis

option to send database object names to the primary Replication Server in the same case as they are stored in Microsoft SQL Server.

Format of origin queue ID

Each record in the transaction log is identified by an origin queue ID that consists of 64 hexadecimal characters (32 bytes). The format of the origin queue ID is determined by the Replication Agent instance, and it varies according to the primary database type.

Table C-1 illustrates the format of the origin queue ID for the Replication Agent for Microsoft SQL Server.

Table C-1: Replication Agent for Microsoft SQL Server origin queue ID

Character	Bytes	Description
0-3	2	database generation ID
4-19	8	transaction max sequence
20-35	8	operation sequence
36-43	2	operation type (begin = 0, data/LOB = 1, commit/rollback = 7FFF
44-59	8	transaction ID
60-63	4	LOB sequence ID

Datatype compatibility

Replication Agent processes Microsoft SQL Server transactions and passes transaction information to the primary Replication Server.

The primary Replication Server uses the datatype formats specified in the replication definition to receive the data from Replication Agent.

The following table describes the default conversion of Microsoft SQL Server datatypes to Sybase Replication Server datatypes.

Table C-2: Microsoft SQL Server to Replication Server default datatype mapping

Microsoft SQL Server datatype	Microsoft SQL Server length/range	Sybase datatype	Sybase length/range	Notes
bit	integer with value of 0 or 1	bit		

Microsoft SQL Server datatype	Microsoft SQL Server length/range	Sybase datatype	Sybase length/range	Notes
int	-2^{31} to $2^{31} - 1$	int		
smallint	integer with value from -2^{15} to $2^{15} - 1$	smallint		
tinyint	integer with value from 0 to 255	tinyint		
decimal	numeric from -10^{38} to $10^{38} - 1$	decimal		
numeric	synonym for decimal datatype	numeric		
money	monetary from -2^{63} to $2^{63} - 1$	money		
smallmoney	monetary from -214,748.3648 to 214,748.3647	smallmoney		
float	floating precision from -1.79E + 308 to 1.79E + 308.			Results in Sybase are machine dependent.
real	floating precision from -3.40E + 38 to 3.40E + 38			Results in Sybase are machine dependent.
datetime	date and time from 01/01/1753 to 12/31/9999	datetime		
smalldatetime	date and time from 01/01/1900 to 06/06/2079	datetime		
timestamp	database-wide unique number	timestamp		To retain the actual value assigned in SQL Server, replicate to the varbinary(8) datatype.
uniqueidentifier	globally unique identifier	varbinary		No Sybase equivalent. Map to binary(38) or varbinary(38) datatype.
char	fixed length up to 8000 characters	char	32K	
varchar	variable length up to 8000 characters	varchar	32K	

Microsoft SQL Server datatype	Microsoft SQL Server length/range	Sybase datatype	Sybase length/range	Notes
text	variable length up to 2^{31} - 1 characters	text	2 GB	
nchar	fixed length Unicode up to 4000 characters	unichar or char	32K	To use the unichar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype. Actual maximum length is @@ncharsize * number of characters.
nvarchar	variable length Unicode up to 4000 characters	univarchar or varchar	32K	To use the univarchar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype. Actual maximum length is @@ncharsize * number of characters.
ntext	variable length Unicode up to 2^{30} - 1 characters	text	2 GB	
binary	fixed length up to 8000 bytes	binary	32K	
varbinary	variable length up to 8000 bytes	varbinary	32K	
image	variable length up to 2^{31} - 1 bytes	image	2 GB	
sql_variant	any datatype except text, ntext, timestamp, and sql_variant, up to 8000 bytes	varbinary	32K	

Microsoft SQL Server datatype restrictions

Replication Server and Replication Agent impose the following constraints on Microsoft SQL Server datatypes:

- If you use a version of Replication Server prior to version 12 with Replication Agent for Microsoft SQL Server, you must set the value of the Replication Agent use_rssd configuration parameter to true to avoid problems with the smallint and tinyint datatypes.
- If you use a version of Replication Server prior to version 12.5:
  - char and varchar dataypes that contain more than 255 bytes will be truncated to 255 bytes.
  - nchar and nvarchar multi-byte character dataypes will be replicated as char or varchar single-byte datatypes.

Replication Agent for Microsoft SQL Server transaction log

Replication Agent uses its own proprietary transaction log to capture and record transactions in the primary database for replication. The Replication Agent transaction log consists of a set of shadow tables, stored procedures, and triggers that are created in the primary database.

The Replication Agent transaction log is created by invoking the pdb_xlog command with the create keyword. When you invoke this command, Replication Agent generates a SQL script that is run in the primary database. This script (stored in the *create.sql* file in the *rax-12_5\inst_name\scripts\xlog* directory) creates the Replication Agent transaction log components referred to as the *transaction log base objects*. The transaction log base objects must be created before any objects can be marked for replication in the primary database.

Note This section describes the schema and details of the Replication Agent transaction log for a Microsoft SQL Server database. For more general information about the Replication Agent transaction log, see Chapter 3, "Administering Sybase Replication Agent," in this book.

Transaction log components

There are three types of Microsoft SQL Server database objects used for transaction log components:

- tables
- stored procedures
- triggers

There are several tables used by the Replication Agent transaction log, both as base objects and as shadow tables for marked primary tables in the primary database. All table components of the transaction log contain at least one index, and some contain more than one index.

Stored procedures are used by the Replication Agent transaction log to capture transactions for replication.

Triggers are used to capture the insert, update, and delete operations in marked primary tables.

Specifying the transaction log object name prefix

Before you create the Replication Agent transaction log base objects, you can specify the object name *prefix* string that will be used to name transaction log objects. You can set this prefix string to avoid conflicts with the names of existing database objects in your primary database.

The value of the pdb_xlog_prefix parameter is the prefix string used in all Replication Agent transaction log component names. Use the ra_config command to change the value of the pdb_xlog_prefix parameter.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log objects in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, Replication Agent will not be able to find the transaction log objects that use the old prefix.

Transaction log object names

There are two variables in the transaction log component database object names shown in this appendix:

- *prefix* — represents the one- to three-character string value of the pdb_xlog_prefix parameter (the default is ra_).
- *xxx* — represents an alphanumeric counter, a string of characters that is (or may be) added to a database object name to make that name unique in the database.

You can use the pdb_xlog command to view the names of Replication Agent transaction log components in the primary database.

Transaction log base tables

Table C-3 lists the tables that are considered Replication Agent transaction log base objects. No permissions are granted on these tables when they are created.

Table C-3: Replication Agent Transaction Log Base Tables

Description	Database Name
transaction log system table	*prefix*xlog_system_
marked objects table	*prefix*marked_objs_*xxx*
transaction log table	*prefix*tran_log_*xxx*
LOB columns table	*prefix*blob_column_*xxx*
exception holding table	*prefix*exception_*xxx*
transaction sequence counter table	*prefix*tran_seq_*xxx*
procedure sequence counter table	*prefix*proc_seq_*xxx*
Log Reader work table	*prefix*lr_reptran_*xxx*
Log Admin work table	*prefix*rawork_*xxx*
transaction active table	*prefix*tranactive_*xxx*
procedure active table	*prefix*procactive_*xxx*

Transaction log stored procedures

Table C-4 lists the stored procedures that are considered Replication Agent transaction log base objects. Execute permission is granted to public when these stored procedures are created.

Note The transaction log stored procedures listed in Table C-4 have no effect when executed outside the context of replication.

Table C-4: Replication Agent Transaction Log Stored Procedures

Procedure	Database Name
transaction information capture	*prefix*get_tx_info_*xxx*
log truncation	*prefix*truncate_logs_*xxx*

Marker procedures and shadow tables

Table C-5 lists the marker procedures and marker shadow tables that are considered Replication Agent transaction log base objects. No permissions are granted when these procedures and tables are created.

Table C-5: Replication Agent Marker Procedures and Shadow Tables

Procedure/Table	Database Name
transaction log-marker procedure	rs_marker*xxx*
transaction log-marker shadow table	*prefix*markersh_*xxx*
dump marker procedure	rs_dump*xxx*
dump marker shadow table	*prefix*dumpsh_*xxx*

Transaction log objects for each marked table

Table C-6 lists the Replication Agent transaction log objects that are created for each primary table that is marked for replication. These objects are created only when a table is marked for replication. These objects are not considered transaction log base objects.

Note The transaction log stored procedures listed in Table C-6 have no effect when executed outside the context of replication.

Table C-6: Replication Agent Transaction Log Objects for Each Marked Table

Marked Table Object	Database Name
shadow table	*prefix*sh_*xxx*
shadow row procedure	*prefix*srp_*xxx* (public execute permission)
LOB shadow table	*prefix*bsh_*xxx*
LOB shadow row procedure	*prefix*bsrp_*xxx* (public execute permission)
insert trigger	*prefix*instrg_*xxx* (or existing-trigger name)
update trigger	*prefix*updtrg_*xxx* (or existing-trigger name)
delete trigger	*prefix*deltrg_*xxx* (or existing-trigger name)

Note The marked objects table contains the mapping between marked objects and their corresponding shadow tables and shadow row procedures.

Reserved names

When a primary table is marked for replication, Replication Agent creates a shadow table to record the replicated operations. The shadow table has all the columns of the primary table, plus three columns that the Replication Agent creates for its use:

- ra_tran_id_
- ra_opid_
- ra_img_type_

Note These column names are fixed, not generated, therefore the "ra_" portion of the column name does not change when the value of the pdb_xlog_prefix parameter changes.

If a primary table has a column with the same name as one of the Replication Agent shadow table columns, the table marking procedure fails. Replication Agent flags the conflicting column in the table-marking script and the pdb_setreptable command returns an error.

In this event, you must change the name of the conflicting column in the primary table in order to replicate changes to that table. After you change the name of the conflicting column, you can modify the table-marking script and run it manually to mark the primary table.

Note If you want to use the primary table's original column name in the replicate table, you can create a function string in the replicate Replication Server to map the new column name in the primary table to a different column name in the replicate table.

Marked table triggers

If a trigger exists on a primary table when that table is marked for replication, the existing trigger is modified to support Sybase replication.

The Sybase replication action in a trigger begins and ends with comments that serve as markers so that, when the table is unmarked, the Sybase replication action can be removed from the trigger.

Note Do not remove or alter the Sybase replication comments in triggers.

Microsoft SQL Server 7.0 and Microsoft SQL Server 2000 allow multiple triggers of each type to be created on a table. If more than one trigger of a given type exists on a table that you mark for replication, the first trigger is the one modified for Sybase replication.

Getting actual names of the transaction log objects

Because Sybase Replication Agent generates the names of its transaction log objects using its own algorithms, there is not any obvious correlation between the name of a primary object and the names of the transaction log objects (such as shadow tables) that record replicated operations for the primary object. If you want to find out the names of transaction log objects associated with a primary object, use the following procedure.

❖ **To find out the names of transaction log objects associated with a primary object:**

1 At the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

```
pdb_xlog
```

The pdb_xlog command returns a list of the transaction log base objects.

2 At the Replication Agent administration port, invoke the pdb_setrepproc command with no keywords:

```
pdb_setrepproc
```

The pdb_setrepproc command returns a list of all stored procedures marked for replication, including all the Replication Agent transaction log objects associated with each marked procedure.

3 At the Replication Agent administration port, invoke the pdb_setreptable command with no keywords:

```
pdb_setreptable
```

The pdb_setreptable command returns a list of all tables marked for replication, including all the Replication Agent transaction log objects associated with each marked table.

Marked objects table

One of the Replication Agent transaction log base objects is the **marked objects table**. The marked objects table contains an entry for each marked object in the primary database (both tables and stored procedures). Each marked object entry contains the following information:

- the name of the marked primary object
- the primary object's replicated name (if any)
- the type of the primary object
- the "replication enabled" flag for the primary object
- the name of the shadow table for the primary object
- the name of the operation-image procedure
- the name of the LOB shadow table (if the primary object is a LOB column)
- the name of the LOB image procedure (if the primary object is a LOB column)
- the owner of the primary object
- the "send owner" flag

Administering the transaction log

The only transaction log administration required is backing up the transaction log and truncation.

Backing up and restoring the transaction log

Sybase Replication Agent does not support backing up and restoring the transaction log automatically.

Sybase recommends that you use the database backup utilities provided with your Microsoft SQL Server software to periodically back up the transaction log.

Note Sybase Replication Agent does not support replaying transactions from a restored log.

Truncating the transaction log

Sybase Replication Agent provides features for both automatic and manual log truncation.

Replication Agent provides two options for automatic transaction log truncation:

- Periodic truncation, based on a time interval you specify.
- Automatic truncation whenever Replication Agent receives a new LTM Locator value from the primary Replication Server.

You also have the option to switch off automatic log truncation. By default, automatic log truncation is switched off.

You can specify the automatic truncation option you want (including none) by using the ra_config command to set the value of the truncation_type configuration parameter.

If you want to truncate the transaction log automatically based on a time interval, use the ra_config command to set the value of the truncation_interval configuration parameter.

You can truncate the Replication Agent transaction log manually, at any time, by invoking the pdb_truncate_xlog command at the Replication Agent administration port.

If you want to truncate the transaction log at a specific time, you can use a scheduler utility to execute the pdb_truncate_xlog command automatically.

Replication Agent for Microsoft SQL Server setup test scripts

Sybase Replication Agent provides a set of test scripts that automate the process of creating a replication test environment that you can use to verify the installation and configuration of the Replication Agent software and the basic function of the other components in your replication system.

The Replication Agent test scripts are located in the *scripts* subdirectory under the Replication Agent base directory. For example: *rax-12_5\scripts*.

The Replication Agent test scripts perform the following tasks:

1 Create a primary table.

2 Create a replicate table that corresponds to the primary table.

3 Create the primary database server connection in the primary Replication Server.

4 Create the replication definition in the primary Replication Server.

5 Test the replication definition.

6 Create the subscription in the replicate Replication Server.

7 Test the subscription.

8 Create the Replication Agent transaction log in the primary database.

9 Modify the primary table.

10 Clean up and remove the replication test environment created by the other scripts.

Before you begin

Before running the test scripts, make sure that you have:

- Installed a Microsoft SQL Server database server
- Installed a database server to act as a replicate data server
- Created the replicate database in the replicate data server
- Installed primary and replicate Replication Servers (PRS and RRS)

Note The PRS and RRS can be the same Replication Server. You must create routes between them if the PRS and RRS are not the same Replication Server.

- Added the replicate database as an RRS-managed database
- Installed the Replication Agent software, created a Replication Agent instance, and used the ra_config command to set the following configuration parameters:
  - ra_config ltl_character_case, lower
  - ra_config rs_source_ds, rax
  - ra_config rs_source_db, test

Note These recommended configuration parameter values are for this test only. The values you should use in a production environment (or even a different test environment) may be different.

- Configured all the pds, rs, and rssd connection parameters and tested them successfully with the Replication Agent test_connection command
- Confirmed that all servers are running
- Confirmed that the Replication Agent instance is in the *Admin* state

A word about the isql tool

The database access tool provided with Microsoft SQL Server is Microsoft isql. You must use Microsoft isql (or a compatible tool) to access the Microsoft SQL Server database to execute some of the test scripts documented in this Appendix.

Although the name of the Microsoft isql tool is the same as the Sybase tool called isql, the Sybase and Microsoft tools are not compatible. For example, you cannot use the Sybase isql tool to access the Microsoft SQL Server data server and you cannot use the Microsoft isql tool to access the Replication Agent administration port.

If you have both Sybase and Microsoft isql tools loaded on the same computer, you may need to change an environment variable (possibly the *path* variable) to avoid problems when you invoke one of the isql tools.

Create the primary table

Use your Microsoft SQL Server database access tool (such as Microsoft isql) to log in to the Microsoft SQL Server data server and execute the mssql_create_test_primary_table.sql script to create the primary table in the primary database.

Note This script (mssql_create_test_primary_table.sql) does not specify a database name. You must either edit the script to include a use command, or invoke isql with the -d option.

Create the replicate table

Use isql or another query processor to log in to the replicate database server and execute the ase_create_test_replicate_table.sql script to create the replicate table in the replicate database.

Create the Replication Server connection for Replication Agent

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_connection.sql script to create the Replication Server connection to the primary database.

Create the replication definition

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_repdef.sql script to create a test replication definition.

Test the replication definition

Use isql or another query processor to log in to the RSSD of the primary Replication Server and execute the rssd_helprep_test_repdef.sql script to test the replication definition.

Create the subscription

There are two scripts provided for this step:

- rs_create_test_sub.sql — designed for use with Replication Server version 11.5 or later
- rs_create_test_sub_for_11.0.x.sql — designed for use with Replication Server version 11.0.x

❖ **To create the test subscription:**

1 Edit the appropriate script file (rs_create_test_sub.sql or rs_create_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that

you used to create the connection to the replicate data server and replicate database. These values are initially set to ase.test.

2 Use isql or another query processor to access the replicate Replication Server and execute the appropriate script to create the test subscription.

Note If you are using the rs_create_test_sub_for_11.0.x.sql script, the script must be executed twice because, even though the PRS and RRS are the same Replication Server, there is not enough time between the command executions in the script to allow Replication Server to complete all its tasks before the next script command is executed. For this reason, you may wish to execute the commands in this script separately.

Note This procedure assumes that no materialization is necessary. See Appendix E, "Materializing Subscriptions to Primary Data," for more information about materialization.

Test the subscription

Use isql or another query processor to access the RSSD of the replicate Replication Server and execute the rssd_helpsub_test_sub.sql script to test the subscription.

Create the Replication Agent transaction log

If you have not already created the Replication Agent transaction log in the primary database, create the transaction log now.

❖ **To create the Replication Agent transaction log:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to create the transaction log.

```
pdb_xlog create
```

See "Creating the Replication Agent transaction log" on page 16 for more information.

Mark a primary table for replication

Mark the test primary table (rax_test) that was created in the Microsoft SQL Server database by the mssql_create_test_primary_table.sql script.

❖ **To mark the test primary table for replication:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to mark the rax_test table for replication.

```
pdb_setreptable rax_test, mark
```

See "Marking objects in the primary database" on page 18 for more information.

Note Make sure that replication is enabled for the rax_test table. See "Enabling and disabling replication for marked tables" on page 61 for more information.

Start replication

If you have not already put the Replication Agent instance in the *Replicating* state, put the Replication Agent instance in the *Replicating* state now.

❖ **To start test replication:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to put the Replication Agent instance in the *Replicating* state.

```
resume
```

See "Starting replication" on page 24 for information.

Execute the test transaction script in Microsoft SQL Server

Use your Microsoft SQL Server database access tool (such as Microsoft isql) to log in to the Microsoft SQL Server data server and execute the mssql_primary_test_transactions.sql script to generate transactions in the primary table in the primary database.

Note This script (mssql_primary_test_transactions.sql) does not specify a database name. You must either edit the script to include a use command, or invoke isql with the -d option.

Check results of replication

Use your Microsoft SQL Server database access tool (such as Microsoft isql) to log in to the Microsoft SQL Server data server and execute the mssql_select_test_primary.sql script to view the changes in the test primary table in the primary database.

Note This script (mssql_select_test_primary.sql) does not specify a database name. You must either edit the script to include a use command, or invoke isql with the -d option.

Use isql or another query processor to log in to the replicate database and execute the ase_select_test_replicate.sql script to verify that the transactions generated in the primary database were replicated to the test replicate table in the replicate database.

Clean up the test environment

Use the following procedure to clean up and remove the replication test environment that you created in the previous sections.

❖ **To clean up and remove the replication test environment:**

1 Log into the Replication Agent administration port using isql (or another query processor).

2 Use the following command to quiesce the Replication Agent instance.

```
quiesce
```

3 Use the following command to remove the Replication Agent transaction log and unmark the test primary table (rax_test) in the Microsoft SQL Server database.

```
pdb_xlog remove, force
```

4 Log into the replicate Replication Server and execute the following scripts:

   - rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql (to drop the test subscription)

   Edit the appropriate script file (rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used in the rs_create_test_sub.sql script. These values are initially set to ase.test.

   - rs_drop_test_repdef.sql (to drop the test replication definition)
   - rs_drop_test_connection.sql (to drop the test connection to the primary database)

5 Log into the replicate data server and execute the ase_drop_test_replicate_table.sql script to drop the test replicate table.

6 Use your Microsoft SQL Server database access tool (such as Microsoft isql) to log in to the Microsoft SQL Server data server and execute the mssql_drop_test_primary_table.sql script to drop the test primary table.

Note This script (mssql_drop_test_primary_table.sql) does not specify a database name. You must either edit the script to include a use command, or invoke isql with the -d option.

APPENDIX D

Administering the Replication Agent for Oracle

The term “Replication Agent for Oracle” refers to an instance of the Sybase Replication Agent version 12.5 software installed and configured for a primary database that resides on an Oracle database server. This appendix describes the characteristics of the Sybase Replication Agent that are unique to the Replication Agent for Oracle implementation.

Note For information on the basic functionality of Sybase Replication Agent version 12.5, refer to the other parts of this *Sybase Replication Agent Administration Guide*.

This appendix includes the following sections:

- Oracle-specific issues
- Replication Agent for Oracle transaction log
- Replication Agent for Oracle setup test scripts

Oracle-specific issues

This section describes general issues and considerations that are specific to using Sybase Replication Agent version 12.5 with the Oracle database server.

The following topics are included in this section:

- Replication Agent connectivity
- Replication Agent permissions
- Parameter length in stored procedures
- WHEN clause in triggers
- Multiple triggers

- Configuring and tuning Replication Agent for Oracle
- Maximum number of columns in a table
- Character case of database object names
- Format of origin queue ID
- Datatype compatibility

Replication Agent connectivity

Connectivity between the Replication Agent for Oracle and the Oracle data server is by way of the Oracle JDBC thin driver, version 8.1.6.

The Oracle JDBC driver must be installed on the Replication Agent host machine and the directory this driver is installed in must be in the *CLASSPATH* environment variable.

The TNS Listener Service must be installed and running on the primary database so that the Replication Agent instance can connect to it. See the Oracle 8 Networking document for more information.

Replication Agent permissions

Sybase Replication Agent requires that all user IDs that create transactions that will be replicated must have select authority on the following Oracle dynamic performance views:

- V_$SESSION
- V_$TRANSACTION

To set up select authority for the Replication Agent, you must log in as the SYS user (or a user with SYSDBA ROLE) and execute the following SQL commands:

```
grant select on V_$SESSION to public
grant select on V_$TRANSACTION to public
```

Parameter length in stored procedures

The Oracle database server allows you to create stored procedures specifying only the datatype of each parameter. Parameter length specification is optional.

To support its logging process, Sybase Replication Agent requires a length specification for all stored procedure parameters. Replication Agent accommodates procedure parameters that do not have a length specified by using a "hard-coded" length, based on the datatype specified for the parameter. If this "default" parameter length is not sufficient for your stored procedure replication needs, you can alter the Oracle procedure definition to include a length specification for the parameter.

WHEN clause in triggers

The Replication Agent table-marking script creates (or alters existing) triggers on a table. In the case of existing triggers, the script inserts Replication Agent code at the end of the trigger, so that any existing trigger operations take place first.

If a trigger contains a WHEN clause that could cause an early exit from the trigger, this jeopardizes the reliability of the Replication Agent trigger code.

Note Sybase recommends that you do not use a WHEN clause that can cause an early exit from the trigger on Oracle tables you mark for replication.

Multiple triggers

Oracle allows you to create multiple AFTER row triggers that fire for the same statement on the same table. However, the order in which these triggers fire is indeterminate.

If you have more than one AFTER row trigger on a table, Replication Agent might not be able to create its data capture triggers, or if created, the triggers might not work correctly.

To avoid this problem, do not use more than one AFTER row trigger on a table you mark for replication.

Configuring and tuning Replication Agent for Oracle

Sybase makes recommendations on the following subjects for tuning Replication Agent for Oracle:

- Oracle sequence caching
- Transaction log tablespace
- NLS environment variables

Oracle sequence caching

Oracle supports sequence "caching." By default, when Replication Agent transaction sequence objects are created, they are created with a cache value of 100. You can alter the sequence cache value to meet the needs of your environment. See your Oracle documentation for more information about tuning sequences.

Transaction log tablespace

The value of the Replication Agent pdb_xlog_device configuration parameter specifies the Oracle tablespace where Replication Agent transaction log objects are created. If no value is specified for the pdb_xlog_device parameter, transaction log objects are created either on the "system" tablespace or on the default tablespace assigned to the Replication Agent user ID that has owner permission in the Oracle data server.

NLS environment variables

If the primary database resides on an Oracle version 8.1.6 server, you must set the *NLS_DATE_FORMAT* variable for the Oracle instance. You can set the *NLS_DATE_FORMAT* variable to any value, but the variable must be set. In Oracle version 8.1.5 or earlier, you need not set the *NLS_DATE_FORMAT* variable.

Maximum number of columns in a table

A table marked for replication can have no more than three columns less than the maximum number allowed by the Microsoft SQL Server database.

For example, if the database allows up to 232 columns in a table, any table marked for replication can have up to 229 columns. Replication Agent adds three additional columns to a transaction log shadow table, which must not exceed the maximum number of columns allowed by the database.

Character case of database object names

Database object names must be delivered to the primary Replication Server in the same format as they are specified in replication definitions. For example, if a replication definition specifies a table name in all lowercase, then that table name must appear in all lowercase when it is sent to the primary Replication Server by the Replication Agent.

Note Replication will fail if a database object name is delivered to the primary Replication Server in a format different from that specified in the replication definition.

Sybase Replication Agent version 12.5 gives you some control over the way it treats the character case of database object names when it sends LTL to the primary Replication Server. You have three options:

- as-is — database object names are passed to Replication Server in the same format as they are actually stored in the primary database server
- lower — database object names are passed to Replication Server in *all lowercase*, regardless of the way they are actually stored in the primary database server
- upper — database object names are passed to Replication Server in *all uppercase*, regardless of the way they are actually stored in the primary database server

You specify the character case option you want by setting the value of the ltl_character_case configuration parameter. The parameter values are asis, lower, and upper. The default value of the ltl_character_case parameter is asis.

In the case of the Oracle database server, database object names are stored in all uppercase. Therefore, the asis and upper options have the same affect on database object names sent from Replication Agent for Oracle to the primary Replication Server.

Format of origin queue ID

Each record in the transaction log is identified by an origin queue ID that consists of 64 hexadecimal characters (32 bytes). The format of the origin queue ID is determined by the Replication Agent instance, and it varies according to the primary database type.

Table D-1 illustrates the format of the origin queue ID for the Replication Agent for Oracle.

Table D-1: Replication Agent for Oracle origin queue ID

Character	Bytes	Description
0-3	2	database generation ID
4-19	8	transaction max sequence
20-35	8	operation sequence
36-43	2	operation type (begin = 0, data/LOB = 1, commit/rollback = 7FFF
44-59	8	transaction ID
60-63	4	LOB sequence ID

Datatype compatibility

Replication Agent for Oracle processes Oracle transactions and passes data to the primary Replication Server.

The primary Replication Server uses the datatype formats specified in the replication definition to receive the data from Replication Agent for Oracle.

Table D-2 describes the conversion of Oracle datatypes to Sybase datatypes.

Table D-2: Oracle to Sybase datatype mapping

Oracle datatype	Oracle length/range	Sybase datatype	Sybase length/range	Notes
CHAR	255 bytes	char	32K	
DATE	7 bytes, fixed-length, default format: DD-MM-YY	datetime	8 bytes	Replication Server supports dates from January 1, 1753 to December 31, 9999. Oracle supports dates from January 1, 4712 BC to December 31, 4712 AD.
LONG	2 GB, variable-length character data	text	2GB	
LONG RAW	2 GB, variable-length binary data	image	2GB	
BLOB	4 GB, variable-length binary large object	image	2GB	Values over 2GB are truncated.
CLOB	4 GB, variable-length character large object	text	2GB	Values over 2GB are truncated.
NCHAR	255 bytes, multi-byte characters	unichar or char	32K	To use the unichar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
NCLOB	4 GB, variable-length multi-byte character large object	text	2GB	Values over 2GB are truncated.
NVARCHAR2	2000 bytes, variable-length, multi-byte character data	univarchar or varchar	32K	To use the univarchar datatype, the use_rssd parameter must be true and the replication definition must specify a Unicode datatype.
BFILE	4 GB, locator points to large binary file	image	2GB	Values over 2GB are truncated.
MLSLABEL	5 bytes, variable-length binary OS label			Not supported.

Oracle datatype	Oracle length/range	Sybase datatype	Sybase length/range	Notes
NUMBER (p,s)	21 bytes, variable-length numeric data	float, int, real, number, or decimal	float is 4 or 8 bytes. int is 4 bytes. real is 4 bytes. number and decimal are 2 to 17 bytes.	The float datatype can convert to scientific notation if the range is exceeded. Integers (int) are truncated if they exceed the Replication Server range of 2,147,483,647 to -2,147,483,648 or $1x10^{-130}$ to $9.99x10^{25}$. The number and decimal datatypes are truncated if they exceed the range of -10^{38} to $10^{38}-1$. Oracle precision ranges from 1 to 38 digits. Default precision is 18 digits. Oracle scale ranges from -84 to 127. Default scale is 0.
RAW	2000 bytes, variable-length binary data	binary or varbinary	32K	
ROWID	6 bytes, binary data representing row addresses	char	32K	
VARCHAR2	2000 bytes, variable-length character data	varchar	32K	

Oracle datatype restrictions

Replication Server and Replication Agent impose the following constraints on Oracle datatypes:

- Oracle BLOB, CLOB, NCLOB, and BFILE datatypes that contain more than 2GB will be truncated to 2GB.
- If you use a version of Replication Server prior to version 12.5:
  - CHAR, RAW, ROWID, and VARCHAR2 datatypes that contain more than 255 bytes will be truncated to 255 bytes.
  - Oracle NCHAR and NVARCHAR2 multi-byte character datatypes will be replicated as char or varchar single-byte datatypes.
- Oracle NUMBER datatype in the *integer* representation:

- The corresponding Replication Server int datatype has a smaller absolute maximum value.
- The Oracle NUMBER absolute maximum value is 38 digits of precision, between 9.9 x 10^{125} and 1 x 10^{-130}. The Sybase int value is between 2^{31} - 1 and -2^{31} (2,147,483,647 and -2,147,483,648), inclusive.
- Oracle NUMBER values greater than the Sybase maximum are rejected by Replication Server.

- Oracle NUMBER datatype in the *floating point* representation:
  - The precision has the same limitations in range as the integer representation.
  - If the floating point value is outside the Sybase range of 2^{31} - 1 and -2^{31} (2,147,483,647 and -2,147,483,648), Replication Agent for Oracle converts the number into exponential format to make it acceptable to Replication Server. No loss of precision or scale occurs.

See Also

Replication Server *Reference Manual* for more information on replication definitions and the create replication definition command.

Replication Agent for Oracle transaction log

Sybase Replication Agent uses its own proprietary transaction log to capture and record transactions in the primary database for replication. The Replication Agent transaction log consists of a set of shadow tables, stored procedures, and triggers that are created in the primary database.

The Replication Agent transaction log is created by invoking the pdb_xlog command with the create keyword. When you invoke this command, Replication Agent generates a SQL script that is run in the primary database. This script (stored in the *create.sql* file in the *rax-12_5\inst_name\scripts\xlog* directory) creates the Replication Agent transaction log components referred to as the *transaction log base objects*. The transaction log base objects must be created before any objects can be marked for replication in the primary database.

Note This section describes the schema and details of the Replication Agent transaction log for an Oracle database. For more general information about the Replication Agent transaction log, see Chapter 3, "Administering Sybase Replication Agent," in this book.

Transaction log components

There are six types of Oracle database objects used for Replication Agent transaction log components:

- tables
- stored procedures
- triggers
- indexes
- sequences
- synonyms

The Replication Agent transaction log uses several tables, both as base objects and as shadow tables for marked primary tables and procedures. All table components of the transaction log contain at least one index. Some contain more than one index. Public synonyms are created for tables and sequences.

Stored procedures are used by the Replication Agent transaction log to capture transactions for replication, convert the DATE data type, and perform various other log operations.

Triggers are used to capture the insert, update, and delete operations in marked primary tables.

Sequences are used to assign unique operation and transaction numbers to the primary operations recorded in the transaction log.

Transaction log object names

There are two variables in the transaction log component database object names shown in this appendix:

- prefix — represents the one- to three-character string value of the pdb_xlog_prefix parameter (the default is ra_).

- xxx — represents an alphanumeric counter, a string of characters that is (or may be) added to a database object name to make that name unique in the database.

The value of the pdb_xlog_prefix parameter is the prefix string used in all Replication Agent transaction log component names.

If this value conflicts with the names of existing database objects in your primary database, you can change the value of the pdb_xlog_prefix parameter by using the ra_config command.

The value of the pdb_xlog_prefix_chars parameter is a list of the non-alphanumeric characters allowed in the prefix string specified by pdb_xlog_prefix. This list of allowed characters is database-specific. For example, in Oracle the only non-alphanumeric characters allowed in a database object name are the $, #, and _ characters.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log objects in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, Replication Agent will not be able to find the transaction log objects that use the old prefix.

You can use the pdb_xlog command to view the names of Replication Agent transaction log components in the primary database.

Transaction log base tables

Table D-3 lists the tables that are considered Replication Agent transaction log base objects. No permissions are granted on these tables when they are created.

Table D-3: Replication Agent transaction log base tables

Table	Database Name
transaction log system table	*prefix*XLOG_SYSTEM_
marked objects table	*prefix*MARKED_OBJS_[*xxx*]
transaction log table	*prefix*TRAN_LOG_[*xxx*]
LOB columns table	*prefix*BLOB_COLUMNS_[*xxx*]

Transaction log stored procedures

Table D-4 lists the stored procedures that are considered Replication Agent transaction log base objects. Execute permission is granted to public at the time these stored procedures are created.

Note The transaction log stored procedures listed in Table D-4 have no effect when executed outside the context of replication.

Table D-4: Replication Agent transaction log stored procedures

Procedure	Database Name
transaction information capture	*prefix*GET_TX_INFO_[*xxx*]
log truncation	*prefix*TRUNCATE_LOGS_[*xxx*]

Marker procedures and shadow tables

Table D-5 lists the marker procedures and marker shadow tables that are considered Replication Agent transaction log base objects. No permissions are granted when these procedures and tables are created.

Table D-5: Replication Agent marker procedures and shadow tables

Procedure/Table	Database Name
transaction log marker procedure	RS_MARKER[*xxx*]
transaction log marker shadow table	*prefix*SH_RS_MARKER_[*xxx*]
dump marker procedure	RS_DUMP[*xxx*]
dump marker shadow table	*prefix*SH_RS_DUMP_[*xxx*]

Sequences

Table D-6 lists the Oracle sequences that are considered Replication Agent transaction log base objects.

Table D-6: Replication Agent sequences

Sequence	Database Name
assign operation ID	*prefix*OPIDSEQ_
assign transaction ID	*prefix*GET_TRAN_
assign process ID	*prefix*PROC_SEQ_

Transaction log objects for each marked table

Table D-7 lists the Replication Agent transaction log objects that are created for each primary table that is marked for replication.

These objects are created only when a table is marked for replication, and thus, they are not considered to be transaction log base objects.

Note The transaction log stored procedures listed in Table D-7 have no effect when executed outside the context of replication.

Table D-7: Replication Agent transaction log objects for each marked table

Object	Database Name
shadow table	*prefix*SH_*xxx* (no permissions granted)
LOB shadow table	*prefix*BSH_*xxx* (no permissions granted)
shadow row procedure	*prefix*SRP_*xxx* (execute permission granted to public)
LOB shadow row procedure	*prefix*BSRP_*xxx* (execute permission granted to public)
insert trigger	*prefix*INSTRG_*xxx* (or existing-trigger name)
update trigger	*prefix*UPDTRG_*xxx* (or existing-trigger name)
delete trigger	*prefix*DELTRG_*xxx* (or existing-trigger name)

Note The marked objects table contains the mapping between marked objects and their corresponding shadow tables and shadow row procedures.

Reserved names

When a primary table is marked for replication, Replication Agent creates a shadow table to record the replicated operations. The shadow table has all the columns of the primary table, plus three special columns that Replication Agent creates for its use:

- RA_TRAN_ID_
- RA_OP_ID_
- RA_IMG_TYPE_

Note These column names are fixed, not generated, therefore the "RA_" portion of the variable name does not change when the value of the pdb_xlog_prefix parameter changes.

If a primary table has a column with the same name as a Replication Agent shadow table special column, the table marking procedure fails. Replication Agent flags the conflicting column in the table-marking script and the pdb_setreptable command returns an error.

In this event, you must change the name of the conflicting column in the primary table.

You cannot use these names as names for replicate columns in the replicate database, unless you create function strings to modify the replicated transactions before they are applied to the replicate database.

If you elect to change the name of the conflicting column in the primary table, you can modify the table-marking script and run it manually to mark the primary table.

If you want to use the original column name in the replicate table, you can create a function string in the replicate Replication Server to map the new column name in the primary table to the original column name in the replicate table.

Marked table triggers

If an AFTER row trigger exists on a primary table when that table is marked for replication, the existing trigger is modified to contain a section of code to support Sybase replication.

The Sybase replication section in an existing trigger begins and ends with comments that serve as markers so that, when the table is unmarked, the Sybase replication section can be removed from the trigger.

Note Do not remove or alter the Sybase replication comments in triggers.

Getting actual names of the transaction log objects

Because Sybase Replication Agent generates the names of its transaction log objects using its own algorithms, there is not any obvious correlation between the name of a primary object and the names of the transaction log objects (such

as shadow tables) that record replicated operations for the primary object. If you want to find out the names of transaction log objects associated with a primary object, use the following procedure.

❖ **To find out the names of transaction log objects associated with a primary object:**

1 At the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

```
pdb_xlog
```

The pdb_xlog command returns a list of the transaction log base objects.

2 At the Replication Agent administration port, invoke the pdb_setrepproc command with no keywords:

```
pdb_setrepproc
```

The pdb_setrepproc command returns a list of all stored procedures marked for replication, including all the Replication Agent transaction log objects associated with each marked procedure.

3 At the Replication Agent administration port, invoke the pdb_setreptable command with no keywords:

```
pdb_setreptable
```

The pdb_setreptable command returns a list of all tables marked for replication, including all the Replication Agent transaction log objects associated with each marked table.

Marked objects table

One of the Replication Agent transaction log base objects is the **marked objects table**. The marked objects table contains an entry for each marked object in the primary database (both tables and stored procedures). Each marked object entry contains the following information:

- the name of the marked primary object
- the primary object's replicated name (if any)
- the type of the primary object
- the "replication enabled" flag for the primary object
- the name of the shadow table for the primary object
- the name of the operation-image procedure

- the name of the LOB shadow table (if the primary object is a LOB column)
- the name of the LOB image procedure (if the primary object is a LOB column)
- the owner of the primary object
- the “send owner” flag

Setting up the transaction log

Before you create the Replication Agent transaction log base objects, you can specify the object name prefix string that will be used to name all the transaction log objects.

The pdb_xlog_prefix configuration parameter stores the prefix string. Use the ra_config command to change the value of the pdb_xlog_prefix parameter.

When the pdb_xlog_prefix parameter is set to use the prefix string you want for the Replication Agent transaction log objects, you can use the pdb_xlog command to create the transaction log base objects in the primary database.

Note Replication Agent uses the value of pdb_xlog_prefix to find its transaction log in the primary database. If you change the value of pdb_xlog_prefix after you create the transaction log, Replication Agent will not be able to find the transaction log objects that use the old prefix.

Administering the transaction log

The only transaction log administration required is backing up the transaction log and truncation.

Backing up and restoring the transaction log

Sybase Replication Agent does not support backing up and restoring the transaction log automatically.

Sybase recommends that you use the database backup utilities provided with your Oracle data server software to periodically back up the Replication Agent transaction log.

Note Sybase Replication Agent does not support replaying transactions from a restored log.

Truncating the transaction log

Sybase Replication Agent provides features for both automatic and manual log truncation.

Replication Agent provides two options for automatic transaction log truncation:

- Periodic truncation, based on a time interval you specify.
- Automatic truncation whenever Replication Agent receives a new LTM Locator value from the primary Replication Server.

You also have the option to switch off automatic log truncation. By default, automatic log truncation is switched off.

You can specify the automatic truncation option you want (including none) by using the ra_config command to set the value of the truncation_type configuration parameter.

If you want to truncate the transaction log automatically based on a time interval, use the ra_config command to set the value of the truncation_interval configuration parameter.

You can truncate the Replication Agent transaction log manually, at any time, by invoking the pdb_truncate_xlog command at the Replication Agent administration port.

If you want to truncate the transaction log at a specific time, you can use a scheduler utility to execute the pdb_truncate_xlog command automatically.

Replication Agent for Oracle setup test scripts

Sybase Replication Agent provides a set of test scripts that automate the process of creating a replication test environment that you can use to verify the installation and configuration of the Replication Agent software and the basic function of the other components in your replication system.

The Replication Agent test scripts are located in the *scripts* subdirectory under the Replication Agent base directory. For example: *rax-12_5\scripts*.

The Replication Agent test scripts perform the following tasks:

1 Create a primary table.

2 Create a replicate table that corresponds to the primary table.

3 Create the primary database server connection in the primary Replication Server.

4 Create the replication definition in the primary Replication Server.

5 Test the replication definition.

6 Create the subscription in the replicate Replication Server.

7 Test the subscription.

8 Create the Replication Agent transaction log in the primary database.

9 Modify the primary table.

10 Clean up and remove the replication test environment created by the other scripts.

Before you begin

Before running the test scripts, make sure that you have:

- Installed an Oracle database server
- Installed a database server to act as a replicate data server
- Created the replicate database in the replicate data server
- Installed primary and replicate Replication Servers (PRS and RRS)

Note The PRS and RRS can be the same Replication Server. You must create routes between them if the PRS and RRS are not the same Replication Server.

- Added the replicate database as an RRS-managed database
- Installed the Replication Agent software, created a Replication Agent instance, and used the ra_config command to set the following configuration parameters:
  - ra_config ltl_character_case, lower
  - ra_config rs_source_ds, rax

- ra_config rs_source_db, test

Note These recommended configuration parameter values are for this test only. The values you should use in a production environment (or even a different test environment) may be different.

- Configured all the pds, rs, and rssd connection parameters and tested them successfully with the Replication Agent test_connection command
- Confirmed that all servers are running
- Confirmed that the Replication Agent instance is in the *Admin* state

Create the primary table

Use your Oracle database access tool (such as sqlplus) to log in to the Oracle data server and execute the oracle_create_test_primary_table.sql script to create the primary table in the primary database.

Create the replicate table

Use isql or another query processor to log in to the replicate database server and execute the ase_create_test_replicate_table.sql script to create the replicate table in the replicate database.

Create the Replication Server connection for Replication Agent

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_connection.sql script to create the Replication Server connection to the primary database.

Create the replication definition

Use isql or another query processor to log in to the primary Replication Server and execute the rs_create_test_repdef.sql script to create a test replication definition.

Test the replication definition

Use isql or another query processor to log in to the RSSD of the primary Replication Server and execute the rssd_helprep_test_repdef.sql script to test the replication definition.

Create the subscription

There are two scripts provided for this step:

- *rs_create_test_sub.sql* — designed for use with Replication Server version 11.5 or later
- *rs_create_test_sub_for_11.0.x.sql* — designed for use with Replication Server version 11.0.x

❖ **To create the test subscription:**

1 Edit the appropriate script file (rs_create_test_sub.sql or rs_create_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used to create the connection to the replicate data server and replicate database. These values are initially set to ase.test.

2 Use isql or another query processor to access the replicate Replication Server and execute the appropriate script to create the test subscription.

Note If you are using the *rs_create_test_sub_for_11.0.x.sql* script, the script must be executed twice because, even though the PRS and RRS are the same Replication Server, there is not enough time between the command executions in the script to allow Replication Server to complete all its tasks before the next script command is executed. For this reason, you may wish to execute the different commands in this script separately.

Note This procedure assumes that no materialization is necessary. See Appendix E, “Materializing Subscriptions to Primary Data,” for more information about materialization.

Test the subscription

Use isql or another query processor to access the RSSD of the replicate Replication Server and execute the rssd_helpsub_test_sub.sql script to test the subscription.

Create the Replication Agent transaction log

If you have not already created the Replication Agent transaction log in the primary database, create the transaction log now.

❖ **To create the Replication Agent transaction log:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to create the transaction log.

```
pdb_xlog create
```

See “Creating the Replication Agent transaction log” on page 16 for more information.

Mark a primary table for replication

Mark the test primary table that was created in the Oracle database by the oracle_create_test_primary_table.sql script (rax_test).

❖ **To mark the test primary table for replication:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to mark the rax_test table for replication.

```
pdb_setreptable rax_test, mark
```

See “Marking objects in the primary database” on page 18 for more information.

Note Make sure that replication is enabled for the rax_test table. See “Enabling and disabling replication for marked tables” on page 61 for more information.

Start replication

If you have not already put the Replication Agent instance in the *Replicating* state, put the Replication Agent instance in the *Replicating* state now.

❖ **To start test replication:**

1 Use isql or another query processor to log in to the Replication Agent administration port.

2 Use the following command to put the Replication Agent instance in the *Replicating* state.

```
resume
```

See “Starting replication” on page 24 for information.

Execute the test transaction script in Oracle

Use your Oracle database access tool (such as sqlplus) to log in to the Oracle data server and execute the *oracle_primary_test_transactions.sql* script to generate transactions in the primary table in the primary database.

Check results of replication

Use your Oracle database access tool (such as sqlplus) to log in to the Oracle data server and execute the *oracle_select_test_primary.sql* script to view the changes in the test primary table in the primary database.

Use isql or another query processor to log in to the replicate database and execute the *ase_select_test_replicate.sql* script to verify that the transactions generated in the primary database were replicated to the test replicate table in the replicate database.

Clean up the test environment

Use the following procedure to clean up and remove the replication test environment that you created in the previous sections.

❖ **To clean up and remove the replication test environment:**

1 Log into the Replication Agent administration port using isql (or another query processor).

2 Use the following command to quiesce the Replication Agent instance.

```
quiesce
```

3 Use the following command to remove the Replication Agent transaction log and unmark the test primary table (rax_test) in the Oracle database.

```
pdb_xlog remove, force
```

4 Log into the replicate Replication Server and execute the following scripts:

- rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql (to drop the test subscription)

Edit the appropriate script file (rs_drop_test_sub.sql or rs_drop_test_sub_for_11.0.x.sql) so that the values for *RDS.RDB* on the with replicate at clause for each command match the *RDS.RDB* values that you used in the rs_create_test_sub.sql script. These values are initially set to ase.test.

- rs_drop_test_repdef.sql (to drop the test replication definition)
- rs_drop_test_connection.sql (to drop the test connection to the primary database)

5 Log into the replicate data server and execute the ase_drop_test_replicate_table.sql script to drop the test replicate table.

6 Use your Oracle database access tool (such as sqlplus) to log in to the Oracle data server and execute the oracle_drop_test_primary_table.sql script to drop the test primary table.

APPENDIX E

Materializing Subscriptions to Primary Data

This appendix introduces the concept of bulk materialization and how to use it to set up replication from primary tables in a primary database. This appendix also describes the process of materializing subscriptions to primary tables in a non-Sybase database.

This appendix includes the following sections:

- Understanding materialization
- Unloading data from a primary database
- Loading data into replicate databases
- Atomic materialization
- Nonatomic materialization

Understanding materialization

Materialization is a process of creating and activating subscriptions and copying data from the primary database to the replicate database, thereby initializing the replicate database. Before you can replicate data from a primary database, you must set up and populate each replicate database so that it is in a state consistent with that of the primary database.

There are two types of subscription materialization supported by the Sybase Replication Server:

- *Bulk materialization* — The process of manually creating and activating a subscription and populating a replicate database using data unload and load utilities outside the control of the replication system.
- *Automatic materialization* — The process of creating a subscription and populating a replicate database using Replication Server commands.

Note Sybase Replication Agent does not support automatic materialization.

See the *Replication Server Administration Guide* for more information on subscription materialization methods.

Bulk materialization overview

Sybase recommends that you use bulk materialization to materialize subscriptions to primary data in a non-Sybase database. When you use bulk materialization, you must coordinate and manually perform the following materialization activities:

- Define, activate, and validate the subscription (or create the subscription without materialization).
- Unload the subscription data at the primary site.
- Move the unloaded data to the replicate database.
- Load data into the replicate tables.
- Resume the database connection from the replicate Replication Server to the replicate database so that the replicate database can receive replicated transactions.
- Resume replication at the Replication Agent instance.

There are two bulk materialization options for subscriptions to primary data in a non-Sybase database:

- Atomic bulk materialization
- Nonatomic bulk materialization

Atomic materialization overview

The atomic bulk materialization procedure consists of the following tasks:

1 Stop updates to the primary table and dump the subscription data from the primary database.

2 In the replicate Replication Server, define the subscription

3 In the primary database, use the rs_marker stored procedure or invoke the Replication Agent ra_marker command to activate the subscription using the with suspension option.

4 Load the subscription data into the replicate table.

5 Resume the database connection from the replicate Replication Server to the replicate database.

6 In the replicate Replication Server, validate the subscription.

Note For a complete step-by-step atomic bulk materialization procedure, see "Atomic bulk materialization procedure" on page 305.

Nonatomic materialization overview

The nonatomic bulk materialization procedure consists of the following tasks:

1 In the replicate Replication Server, use the set autocorrection command.

2 In the replicate Replication Server, define the subscription.

3 In the primary database, use the rs_marker stored procedure or invoke the Replication Agent ra_marker command to activate the subscription using the with suspension option.

4 Dump the subscription data from the primary database.

5 In the primary database, use the rs_marker stored procedure or invoke the Replication Agent ra_marker command to validate the subscription.

6 Load the subscription data into the replicate table.

7 Resume the database connection from the replicate Replication Server to the replicate database.

8 When the subscription becomes valid at all Replication Servers, turn off autocorrection.

Note For a complete step-by-step nonatomic bulk materialization procedure, see "Nonatomic bulk materialization procedure" on page 308.

Unloading data from a primary database

Part of the subscription materialization process is unloading the subscription data from the primary table so it can be loaded into the replicate table. Subscription data is the data in the primary table that is requested by the subscription.

Data unloading utilities are usually provided with the primary database server software. You can use one of the OEM-supplied unloading utilities, or a database unload utility of your choice.

Loading data into replicate databases

Part of the subscription materialization process is loading the subscription data from the primary table into the replicate table.

If you are using Sybase Adaptive Server or Sybase SQL Server as the database server for your replicate database, you can use the Sybase bcp utility to load subscription data into the replicate database.

If you are using a non-Sybase database server as the database server for your replicate database, you can use the load utility of your choice to load subscription data into the replicate database.

See Also

- *Sybase Adaptive Server Enterprise Utility Programs* for more information about using the bcp utility to load subscription data into a replicate database in Sybase Adaptive Server version 11.5 or later.
- *Sybase SQL Server Utility Programs* for more information about using the bcp utility to load subscription data into a replicate database in Sybase SQL Server version 11.0.x.

Atomic materialization

Atomic bulk materialization assumes that all applications updating the primary table can be suspended while a copy of the table is made. This copy will be loaded at the replicate site.

You can use this method to retrieve data from the primary database if you can suspend updates to the primary data.

Preparing for materialization

Before you start an atomic bulk materialization procedure, verify the following:

- The primary table exists and contains data.
- You have a user ID with ownership or select privilege on the primary table (or a column to be replicated in the primary table).
- The replicate table exists and contains the appropriate column(s).
- You successfully configured every Replication Server in your replication system.

- You created the replication definition correctly at the primary Replication Server.
- You successfully created the Replication Agent transaction log in the primary database.
- You marked and enabled replication for the primary table in the primary database.
- You have started the Replication Agent instance and put it in the *Replicating* state.

Atomic bulk materialization procedure

❖ **To perform atomic bulk materialization:**

1 Log into the replicate Replication Server as the System Administrator (sa) using isql:

```
isql -Usa -Psa_password -SRRS_servername
```

where *sa* is the System Administrator user ID, *sa_password* is the password for the System Administrator user ID, and *RRS_servername* is the name of the replicate Replication Server.

2 Define the subscription at the replicate Replication Server using the following syntax:

```
1> define subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> [where search_conditions]
5> go
```

The dataserver.database name must match the name you used for your replicate database.

3 Check the subscription at both the primary and replicate Replication Servers. Use the following command to verify that the subscription status is DEFINED:

```
1> check subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

4 Lock the primary table to prevent primary transaction activity. This prevents updates to the primary table during materialization.

5 Unload the subscription data at the primary site using your site's preferred database unload method to select or dump the data from the primary table.

Note When unloading subscription data from the primary table, make sure you select only the columns specified in the replication definition and the rows specified in the subscription.

6 Activate the subscription using the with suspension option at the replicate Replication Server by using the following syntax:

```
1> activate subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> with suspension
5> go
```

7 Wait for the subscription to become active at both the primary and replicate Replication Servers. Execute the check subscription command at both the primary and replicate Replication Servers to verify that the subscription status is ACTIVE.

When the subscription status is ACTIVE at the replicate Replication Server, the database connection for the replicate database is suspended.

8 Restore the primary table to read-write access (unlock).

9 Load the subscription data into the replicate database using the bcp utility or your site's preferred database load utility.

Note Before loading the subscription data into the replicate table, make sure that any data manipulation that will be performed by Sybase Replication Agent (such as datetime conversion) or performed by Replication Server function strings is applied to the unload file.

10 From the replicate Replication Server, resume the database connection for the replicate database using the following syntax:

```
1> resume connection
2> to dataserver.database
3> go
```

11 Validate the subscription at the replicate Replication Server using the following syntax:

```
1> validate subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

12 Wait for the subscription to become valid at both the primary and replicate Replication Servers, then execute the check subscription command at both the primary and replicate Replication Servers to verify that the status is VALID.

When you complete this procedure, the subscription is created, the replicate data is consistent with the primary data, and replication is in progress.

If replication is not in progress when you complete this procedure, see Chapter 6, "Troubleshooting Sybase Replication Agent."

Nonatomic materialization

Nonatomic bulk materialization assumes applications updating the primary table cannot be suspended while a copy of the table is made. Therefore, a nonatomic materialization requires the use of the Replication Server autocorrection feature to get the replicate database synchronized with the primary database.

Note You cannot use nonatomic materialization if the replicate minimal columns feature is set for the replication definition for the primary table.

Preparing for materialization

Before you start a nonatomic bulk materialization procedure, verify the following:

- The primary table exists and contains data.
- You have a user ID with ownership or select privilege on the primary table (or a column to be replicated in the primary table).
- The replicate table exists and contains the appropriate column(s).
- You successfully configured every Replication Server in your replication system.

- You created the replication definition correctly at the primary Replication Server.
- You successfully created the Replication Agent transaction log in the primary database.
- You marked and enabled replication for the primary table in the primary database.
- You have started the Replication Agent instance and put it in the *Replicating* state.

Nonatomic bulk materialization procedure

❖ **To perform nonatomic bulk materialization:**

1 Log into the replicate Replication Server as the System Administrator (sa) using isql:

```
isql -Usa -Psa_password -SRRS_servername
```

where *sa* is the System Administrator user ID, *sa_password* is the password for the System Administrator user ID, and *RRS_servername* is the name of the replicate Replication Server.

2 Turn on the autocorrection feature at the replicate Replication Server using the following syntax:

```
1> set autocorrection on
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

3 Define the subscription at the replicate Replication Server using the following syntax:

```
1> define subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> with suspension
5> go
```

The dataserver.database name must match the name you used for your replicate database.

4 In the primary database, invoke the rs_marker stored procedure to activate the subscription.

5 Check the subscription at both the primary and replicate Replication Servers. Use the following command to verify that the subscription status is ACTIVE:

```
1> check subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

When the subscription status is ACTIVE at the replicate Replication Server, the database connection for the replicate database is suspended.

6 Unload the subscription data at the primary site using your site's preferred database unload method to select or dump the data from the primary tables.

Note When unloading subscription data from the primary table, make sure you select only the columns specified in the replication definition and the rows specified in the subscription.

---

7 In the primary database, invoke the rs_marker stored procedure to validate the subscription.

8 Wait for the subscription to become valid at both the primary and replicate Replication Servers, then execute the check subscription command at both the primary and replicate Replication Servers to verify that the status is VALID.

9 Load the subscription data into the replicate database using the bcp utility or your site's preferred database load utility.

Note Before loading the subscription data into the replicate table, make sure that any data manipulation that will be performed by Sybase Replication Agent (such as datetime conversion) or performed by Replication Server function strings is applied to the unload file.

---

10 From the replicate Replication Server, resume the database connection for the replicate database using the following syntax:

```
1> resume connection
2> to dataserver.database
3> go
```

11 Wait for the subscription to become valid at both the primary and replicate Replication Servers, then execute the check subscription command at both

the primary and replicate Replication Servers to verify that the status is VALID.

When the subscription's status is VALID at the replicate Replication Server, the replicate database is synchronized with the primary database and you can turn off autocorrection.

12 Turn off the autocorrection feature at the replicate Replication Server using the following syntax:

```
1> set autocorrection off
 2> for replication_definition
3> with replicate at dataserver.database
4> go
```

When you complete this procedure, the subscription is created, the replicate data is consistent with the primary data, and replication is in progress.

If replication is not in progress when you complete this procedure, see Chapter 6, "Troubleshooting Sybase Replication Agent."

See also

- *Replication Server Commands Reference* for information on Replication Command Language (RCL) commands.
- *Replication Server Administration Guide* for information on configuring Replication Servers and materialization methods.

Glossary

This glossary describes Sybase Replication Agent terms used in this book.

Adaptive Server The basic database server in the Sybase Client-Server architecture. It manages multiple databases and multiple users, tracks the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory. See also **Open Server** and **Replication Server**.

atomic materialization A materialization method that copies subscription data from a primary to a replicate database in a single atomic operation. No changes to primary data are allowed until subscription data is captured. Atomic materialization is the default method for the Replication Server create subscription command. See also **nonatomic materialization** and **bulk materialization**.

BCP utility A bulk copy transfer utility that provides the ability to load multiple rows of data into a table in a target database. See also **bulk copy**.

bulk copy An Open Client interface for the high-speed transfer of data between a database table and program variables. It provides an alternative to the use of SQL insert and select commands to transfer data.

bulk materialization A materialization method whereby subscription data in a replicate database is initialized outside of the replication system. Bulk materialization involves a series of commands, starting with define subscription. You can use bulk materialization for subscriptions to table replication definitions or function replication definitions. See also **atomic materialization** and **nonatomic materialization**.

client In client/server systems, the part of the system that sends requests to servers and processes the results of those requests.

client application Software that is responsible for the user interface, including menus, data entry screens, and report formats. See also **client**.

commit An instruction to the DBMS to make permanent the changes requested in a transaction. See also **transaction**. Contrast with **rollback**.

data replication — The process of copying data to remote locations. The copied (replicated) data is then kept synchronized with the primary data. Data replication is distinct from data distribution. Replicated data is stored copies of data at particular sites throughout a system and is not necessarily distributed data. Contrast with **data distribution**. See also **transaction replication**.

data server — A server that provides the functionality necessary to maintain the physical representation of a table in a database. Data servers are usually database servers, but they can also be any data repository with the interface and functionality Replication Server requires.

database — A collection of data with a given structure for accepting, storing, and providing data for multiple users.

database connection — A connection that allows a Replication Server to manage the database and distribute transactions to the database. Each database in a replication system can have only one database connection. See also **Replication Server** and **route**.

datatype — A keyword that identifies the characteristics of stored information on a computer. Some common datatypes are: char, int, smallint, date, time, numeric, and float. Different data servers support different datatypes.

DBMS (database management system) — A computer-based system for defining, creating, manipulating, controlling, managing, and using databases. The database management system can include the user interface for using the database, or it can be a stand-alone database system. Compare with **RDBMS**.

function — A Replication Server object that represents a data server operation such as insert, delete, or begin transaction. Replication Server distributes these operations from primary to replicate databases as functions. Each function consists of a function name and a set of data parameters. When a function is to be applied, Replication Server uses function strings to convert a function to a command or set of commands for a type of database. See also **function string**.

function string — A string that Replication Server uses to map a function and its parameters to a data server API. For the rs_select and rs_select_with_lock functions only, the string contains an input template, used to match function strings with the database command. For all other functions, the string also contains an output template, used to format the database command for the destination data server.

gateway — Connectivity software that allows two or more computer systems with different network architectures to communicate.

ID Server — One Replication Server in a replication system is the ID Server. In addition to performing the usual Replication Server tasks, the ID Server assigns unique ID numbers to every Replication Server and database in the replication system, and maintains version information for the replication system.

inbound queue — A stable queue used to spool messages from a Replication Agent to a Replication Server. The inbound queue is under the control of Replication Server.

interfaces file — A file containing information that Sybase Open Client/Open Server applications need to establish connections to other Open Client/Open Server applications. The interfaces file name varies by platform: interfaces for UNIX platforms, sql.ini for Microsoft Windows NT and Windows 95, and win.ini for Windows 3.x.

isql — An interactive SQL client application that can connect and communicate with any Sybase Open Server application, including SQL Server or Adaptive Server.

Java — An object-oriented programming language developed by Sun Microsystems. A "write once, run anywhere" programming language.

Java Bean — A portable, reusable software component written in the Java programming language.

JDBC (Java Database Connectivity) — A communication protocol for database-independent connectivity between Java clients and data servers.

JDK (Java Development Kit) — A software development environment for writing applets and applications in Java.

JRE (Java Runtime Environment) — The Java Runtime Environment consists of the Java Virtual Machine, the Java Core Classes, and supporting files. See also **Java Virtual Machine**.

Java Virtual Machine — The part of the Java Runtime Environment (JRE) responsible for interpreting Java byte codes. See also **JRE (Java Runtime Environment)**.

LAN (local area network) — A computer network located on the user's premises and covering a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary can be subject to some form of regulation. Contrast with **WAN**.

latency — The time it takes to distribute a transaction from a primary database to a replicate database.

LOB (large object) A type of data element associated with a column in a table that contains extremely large quantities of data (often more than 1 gigabyte).

Log Reader A component of Replication Agent that interacts with the primary data server to capture transactions marked for replication. See also **Log Transfer Interface** and **Log Transfer Manager**.

Log Transfer Interface A component of Replication Agent that interacts with Replication Server. The Log Transfer Interface sends LTL to the primary Replication Server so it can distribute replicated transactions. See also **Log Reader** and **Log Transfer Manager**.

Log Transfer Manager A component of Replication Agent that interacts with other Replication Agent components to control and coordinate Replication Agent operations. See also **Log Reader** and **Log Transfer Interface**.

maintenance user A data server login name that Replication Server uses to maintain replicate data. See also **Replication Server**.

marked objects table A Sybase Replication Agent system table. The marked objects table is a user table in the primary database, created by a script generated by the Sybase Replication Agent. The marked objects table contains a row for each object marked for replication in the primary database. See also **Replication Agent**.

materialization The process of copying data specified by a subscription from a primary database to a replicate database, thereby initializing the replicate table, and activating the subscription so that Replication Server can begin replicating data to the replicate database. See also **atomic materialization**, **bulk materialization**, and **non-atomic materialization**.

nonatomic materialization A materialization method that copies subscription data without a lock on the primary database. Changes to the primary table are allowed during data transfer, which may cause temporary inconsistencies between replicate and primary databases. Nonatomic materialization is an optional method for the Replication Server create subscription command. Contrast with **atomic materialization**. See also **bulk materialization**.

ODBC (Open Data Base Connectivity) A standard communication protocol for clients connecting to database servers.

Open Client A Sybase product that provides customer applications, third-party products, and other Sybase products with the interfaces needed to communicate with Open Server applications.

Open Client application An application written using the Sybase Open Client libraries.

Open Server — A Sybase product that provides the tools and interfaces required to create a custom server.

Open Server application — A custom server built with Sybase Open Server.

outbound queue — A stable queue used to spool messages. The Data Server Interface outbound queue spools messages to a replicate data server. The Replication Server Interface outbound queue spools messages to a replicate Replication Server.

primary data — The version of a set of data in a replicated data system from which replication occurs. With Sybase Replication Agent, primary data is stored in Informix Dynamic Server. See also **Replication Agent** and **Replication Server**.

primary database — The database that contains the tables to be replicated. See also **primary data**.

primary key — The column or columns whose data uniquely identify each row in a table.

primary table — A table used as a source for replication. See also **primary data** and **primary database**.

quiescent — A quiescent replication system is one in which all updates have been propagated to their destinations. Some Replication Server commands or procedures require that you first quiesce the replication system.

In the context of Sybase Replication Agent, when the Replication Agent instance is quiescent, it is in the *Admin* state.

RCL (Replication Command Language) — The commands used to manage information in Replication Server.

RDBMS (relational database management system) — The application that controls relational databases. Compare with **DBMS**. See also **relational database**.

relational database — A collection of data in which data is viewed as being stored in tables consisting of columns (data items) and rows (units of information). Relational databases can be accessed by SQL requests. See also **SQL**.

replicate database — Any database that contains data that is replicated from another database via the replication system. See also **primary database** and **Replication Server**.

replicated table — A table that is maintained by Replication Server, in part or in whole. There is one primary version of the table, which is marked for replication; all other versions are replicated copies. See also **Replication Server**.

Replication Agent — A program or thread that transfers transaction log information representing changes made to primary data from a database server to a Sybase Replication Server for distribution to other (replicate) databases.

replication definition — Usually, a description of a table for which subscriptions can be created. The replication definition, maintained by Replication Server, includes information about the columns to be replicated and the location of the primary version of the table. See also **Replication Server** and **subscription**.

Replication Server — The Sybase server program that maintains replicated data on a LAN and processes data transactions received from other Replication Servers on the same LAN or on a WAN. See also **SQL Server**.

Replication Server System Database (RSSD) — The Adaptive Server or SQL Server database containing a Replication Server's system tables. See also **Replication Server** and **SQL Server**.

replication system — A data processing system where data is replicated in multiple databases to provide remote users with the benefits of local data access. Specifically, a replication system that is based upon Replication Server and includes other components such as Replication Agents and data servers.

rollback — An instruction to a database to back out of the changes requested in a unit of work (called a transaction). Contrast with **commit**. See also **transaction**.

route — A one-way message stream from a primary Replication Server to a replicate Replication Server. Routes carry data modification commands (including those for RSSDs) and replicated functions or stored procedures between Replication Servers. See also **Replication Server**.

SQL (Structured Query Language) — A non-procedural programming language used to process data in a relational database. ANSI SQL is an industry standard. See also **SPL**.

stable queue — A store-and-forward queue in which Replication Server messages destined for a route or database connection are stored. Messages written into the stable queue remain there until they can be delivered to the replicate Replication Server or replicate database. See also **database connection**, **Replication Server**, and **route**.

subscription — A request for Replication Server to maintain a replicated copy of a table, or a set of rows from a table, in a replicate database at a specified location. See also **replication definition** and **Replication Server**.

synchronization — The method that ensures primary and replicate tables are equivalent. For example, if transaction number 100 is successful in the primary database, the 100th transaction in the replicate database should also be successful.

table — In a relational DBMS, a two-dimensional array of data or a named data object that contains a specific number of unordered rows composed of a grouping of columns specific for the table. See also **database**.

transaction — A group that can include zero, one, or many database operations (including inserts, updates, and deletes) that are applied or rejected as a whole. Each SQL statement that modifies data is considered a transaction.

transaction log — Generally, the log of transactions that affect the data managed by a database server.

Sybase Replication Agent creates a transaction log that consists of tables, stored procedures, and triggers in the primary database. The Replication Agent Log Reader component reads the transaction log to identify the changes to marked tables in the primary database. See also Replication Server.

transactional consistency — A condition in which all transactions in the primary database are applied in the replicate database in the same order that they were applied in the primary database.

user-defined function — A Replication Server user-defined function is a function that you create. A user-defined function is a custom-built Replication Server function whose name and parameters exactly match the name and parameters of a replicated stored procedure. See also **function** and **function string**.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W