SYBASE®

Heterogeneous Replication Guide

Replication Server®

15.5

DOCUMENT ID: DC36924-01-1550-01

LAST REVISED: March 2010

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

Contents

About This Book

Replication Server® maintains replicated data at multiple sites on a network. Organizations with geographically distant sites can use Replication Server to create distributed database applications with better performance and data availability than a centralized database system can provide.

This book introduces heterogeneous replication concepts, and it addresses the issues peculiar to heterogeneous replication with Sybase replication technology.

In this book, the term *heterogeneous replication* refers to replication between different types of data servers, for example, replicating from an Oracle database to a Sybase™ Adaptive Server® Enterprise (ASE) database.

This book also addresses the issues involved with replication between non-ASE data servers of the same type, for example, replicating from one Oracle database to another Oracle database.

Audience

The *Replication Server Heterogeneous Replication Guide* is to be used to plan, design, implement, or maintain a Sybase replication system that uses heterogeneous or non-ASE data servers.

If you are new to Replication Server, refer to the *Replication Server Design Guide* for an introduction to basic data replication concepts and Sybase replication technology.

How to use this book

This book is divided into four parts:

- Part 1, “Introduction,” provides an overview of data replication concepts and Sybase replication technology, and it introduces the issues specific to replication systems with heterogeneous or non-ASE data servers. It contains these chapters:
  - Chapter 1, “Replication System Overview,” introduces replication system concepts, with a focus on heterogeneous replication using Sybase replication technology.

    - Chapter 2, “Replication Components Detail,” introduces the Sybase software products that you can use to implement a replication system with heterogeneous or non-ASE data servers.
- Part 2, “Non-ASE Primary Data Server Topics” describes component-specific issues for primary data servers and considerations peculiar to a replication system with heterogeneous or non-ASE primary data servers. It contains these chapters:
    - Chapter 3, “DB2 UDB Primary Data Server Issues for z/OS,” describes the considerations specific to the IBM DB2 UDB primary data server on IBM z/OS in a Sybase replication system.
    - Chapter 4, “DB2 UDB Primary Data Server on UNIX, Windows, and Linux,” describes the considerations specific to the IBM DB2 UDB primary data server on UNIX, Windows, and Linux in a Sybase replication system.
    - Chapter 5, “Microsoft SQL Server Primary Data Server Issues,” describes the considerations specific to the Microsoft SQL Server primary data server in a Sybase replication system.
    - Chapter 6, “Oracle Primary Data Server Issues,” describes the considerations specific to the Oracle primary data server in a Sybase replication system.
- Part 3, “Non-ASE Replicate Data Server Topics,” describes how to set up replicate targets and to maintain a replication system using Replication Server with heterogeneous or non-ASE data servers. It contains the following chapters:
    - Chapter 7, “DB2 UDB for z/OS Replicate Data Server Issues,” describes the considerations specific to the IBM DB2 UDB on IBM z/OS replicate data server in a Sybase replication system.
    - Chapter 8, “DB2 UDB Replicate Data Server Issues for UNIX, Windows, and Linux,” describes the considerations specific to the IBM DB2 UDB on UNIX, Windows, and Linux replicate data server in a Sybase replication system.
    - Chapter 9, “Microsoft SQL Server Replicate Data Server Issues,” describes the considerations specific to the Microsoft SQL Server replicate data server in a Sybase replication system.
    - Chapter 10, “Oracle Replicate Data Server Issues,” describes the considerations specific to the Oracle replicate data server in a Sybase replication system.

    - Chapter 11, “Sybase IQ Replicate Data Server Issues,” provides step-by-step instructions for setting up Replication Server® for replicating data into Sybase IQ. It also describes the considerations specific to the Sybase IQ replicate data server in a Sybase replication system.
- Part 4, “Oracle Replication Topics” describes how to manage heterogeneous warm standby for Oracle and how to resynchronize Oracle database. It contains these chapters:
    - Chapter 12, “Managing Heterogeneous Warm Standby for Oracle,” describes one way to create and manage a warm standby application using Replication Server for Oracle database.
    - Chapter 13, “Resynchronizing Oracle Replicate Databases,” describes the procedures to resynchronize the Oracle database.
- Part 5, “Appendixes,” contain supplemental information:
    - Appendix A, “Datatype Translation and Mapping,” lists the class-level datatype translations for all non-ASE data servers supported by Replication Server. It also lists Replication Server datatype names for non-ASE datatypes.
    - Appendix B, “Materialization Issues,” describes the materialization considerations to keep in mind when implementing a replication system with heterogeneous or non-ASE data servers.
    - Appendix C, “Heterogeneous Database Reconciliation,” describes the considerations involved with reconciling data from different databases in a replication system with heterogeneous or non-ASE data servers.
    - Appendix D, “Troubleshooting Heterogeneous Replication Systems,” describes common problems and troubleshooting procedures for Sybase replication systems with heterogeneous or non-ASE data servers.
    - Appendix E, “Reference Implementation for Oracle to Oracle Replication,” describes how you can quickly set up an Oracle to Oracle reference replication environment.

Related documents

The Replication Server documentation set consists of the following:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Product Manuals at http://www.sybase.com/support/manuals/

- *Installation Guide* for your platform – describes installation and upgrade procedures for all Replication Server and related products.
- *New Features Guide* – describes the new features in Replication Server version 15.5 and the system changes added to support those features.
- *Administration Guide* – contains an introduction to replication systems. This manual includes information and guidelines for creating and managing a replication system, setting up security, recovering from system failures, and improving performance.
- *Configuration Guide* for your platform – describes configuration procedures for all Replication Server and related products.
- *Design Guide* – contains information about designing a replication system and integrating heterogeneous data servers into a replication system.
- *Getting Started with Replication Server* – provides step-by-step instructions for installing and setting up a simple replication system.
- *ASE-to-ASE Replication Quick Start Guide* – provides information for Adaptive Server® Enterprise users who quickly want to set up a Replication Server for replicating data from one Adaptive Server database to another.
- *Reference Manual* – contains the syntax and detailed descriptions of Replication Server commands in the Replication Command Language (RCL); Replication Server system functions; Adaptive Server commands, system procedures, and stored procedures used with Replication Server; Replication Server executable programs; and Replication Server system tables.
- *Heterogeneous Replication Guide* (this book) – describes how to use Replication Server to replicate data between databases supplied by different vendors.
- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Troubleshooting Guide* – contains information to aid in diagnosing and correcting problems in the replication system.

- Replication Manager plug-in help, which contains information about using Sybase Central to manage Replication Server.
- Additional Replication Server Options documents that may be helpful:
  - *Replication Server Options Overview Guide* – describes components used for replicating to and from ASE and non-ASE databases.
  - *Release Bulletin Replication Server Options* Version 15.5 for Linux, Microsoft Windows, and UNIX – describes issues for the Replication Server Options.
  - *Release Bulletin Replication Agent* Version 15.5 for Linux, Microsoft Windows, and UNIX – describes features and issues for Replication Agent™.
  - *Installation Guide Replication Agent* Version 15.5 – describes how to install Replication Agent.
  - *Replication Agent Administration Guide* – describes how to extend the capabilities of Replication Server to replicate from non-ASE primary data servers in a Sybase replication system.
  - *Replication Agent Primary Database Guide* – describes the specific issues related to the non-ASE primary data servers in a Sybase replication system.
  - *Replication Agent Reference Manual* – contains the syntax and detailed descriptions of the Replication Agent commands and configuration parameters.
  - *Enterprise Connect™ Data Access Option for Oracle Server Administration and Users Guide* – describes how to configure ECDA Option for Oracle.
  - *Enterprise Connect Data Access Options Users Guide* for Access Services – describes how to configure a DirectConnect™ access service to extend the capabilities of Replication Server to replicate into non-ASE data servers in a Sybase replication system.
  - *Enterprise Connect Data Access and Mainframe Connect™ Server Administration Guide* – describes how to use a Sybase DirectConnect server.
- Replication Agent for DB2 UDB 15.0 documents that may be helpful:
  - *Release Bulletin Replication Agent for DB2 UDB 15.0 for z/OS* – describes features and issues for Replication Agent™

- *Installation Guide Replication Agent for DB2 UDB 15.0 for z/OS* – describes how to install Replication Agent.
- *New Features Bulletin Replication Agent for DB2 UDB 15.0 for z/OS* – describes the new features in Replication Agent and the system changes added to support those features.
- *User and Troubleshooting Guide Replication Agent for DB2 UDB 15.0 for z/OS* – contains information to aid in diagnosing and correcting problems in the replication system.

Other sources of information

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

- The Getting Started CD is included with your software and contains release bulletins, installation guides in PDF format, and other documents or updated information. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.

  You can also access the documents available on the Getting Started CD from the Sybase Product Manuals Web site.

- The Sybase Product Manuals Web site, which can be accessed using a standard Web browser, includes the Replication Server documents that are not included in the Getting Started CD. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click Partner Certification Report.

3 In the Partner Certification Report filter select a product, platform, and timeframe and then click Go.

4 Click a Partner Certification Report title to display the report.

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

1 Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

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

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

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

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

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

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

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following style conventions are used in this manual:

- In a sample screen display, commands that you should enter exactly as shown are in:

  ```
  this font
  ```

- In a sample screen display, words that you should replace with the appropriate value for your installation are shown in:

  ```
  this font
  ```

- In the regular text of this document, the names of files and directories appear like this:

  /usr/u/sybase

- In the regular text of this document, the names of programs, utilities, procedures, and commands appear like this:

  bcp

The conventions for syntax statements in this manual are as follows:

Table 1: SQL syntax conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords.
variable	Variables, or words that stand for values that you fill in.
{ }	Curly braces indicate that you must choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean that choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you can select only one of the options shown.
,	The comma means you can choose as many of the options shown as you like, separating your choices with commas that you type as part of the command.

Accessibility features

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

Replication Server 15.5 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

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

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

For a Section 508 compliance statement for Replication Server, see Sybase Accessibility at http://www.sybase.com/detail_list?id=52484.

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.

PART 1

Introduction

Chapters in this part introduce some data replication concepts and the Sybase replication technology that supports replication systems with heterogeneous or non-ASE data servers.

- Chapter 1, “Replication System Overview,” introduces basic replication system concepts, with a focus on heterogeneous replication using Sybase replication technology.
- Chapter 2, “Replication Components Detail,” introduces the Sybase software products that enable you to implement a heterogeneous replication system using Sybase replication technology.

CHAPTER 1

Replication System Overview

This chapter introduces a basic replication system, with a focus on heterogeneous replication using Sybase replication technology.

Basic replication system

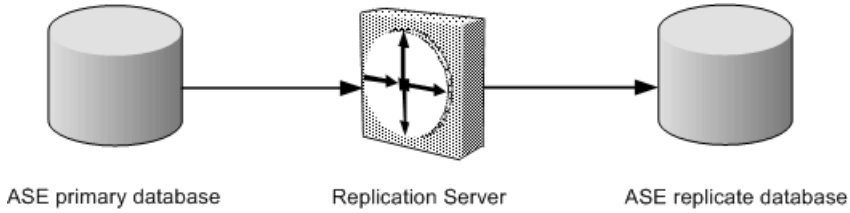
A basic Sybase replication system consists of three components:

- **Primary database** – a database in which original data-changing operations (or transactions) are performed. Only completed transactions are captured for replication.
- **Replication Server** – a Sybase Open Client™ and Open Server™ product that receives transactions to be replicated from a primary database, and delivers them to a replicate database.
- **Replicate database** – a database that receives replicated transactions from a Replication Server and applies those transactions to its own "copy" of the primary data.

If both primary and replicate data servers are Adaptive Server Enterprise (ASE), you can implement a replication system with only these three components. Adaptive Server Enterprise includes all the features necessary to support a Sybase replication system, with no additional components other than the Replication Server.

Figure 1-1 illustrates a basic Sybase replication system, showing the flow of data between two Adaptive Servers and a Replication Server.

Figure 1-1: Basic Sybase replication system



Data flows from the primary database to the Replication Server, and then to the replicate database.

For more information about basic Sybase replication system concepts and Replication Server features, see the first two chapters of the Replication Server *Administration Guide*.

Heterogeneous replication system

The term **heterogeneous replication** refers to replicating data-changing operations between two databases of same or different vendors, except ASE to ASE. For information on ASE to ASE replication, see the *Replication Server Administration Guide Volume 1 and 2*.

Heterogeneous replication includes:

- A replication system in which Adaptive Server Enterprise (ASE) is either the primary or the replicate data server, and a non-ASE data server (such as IBM DB2 UDB) is the other data server.
- A replication system in which the primary and replicate data servers are both non-ASE data servers (for example, Oracle is the primary data server and IBM DB2 UDB is the replicate data server, or Microsoft SQL Server is the primary server and Microsoft SQL Server is the replicate server).

Adaptive Server Enterprise was enhanced to support Replication Server. All of the data server elements required to support Replication Server (that is, a data-change capture mechanism in the primary database, and system tables and stored procedures in the replicate database) are either built into Adaptive Server Enterprise or enabled by utilities that are provided with the Replication Server or Adaptive Server software.

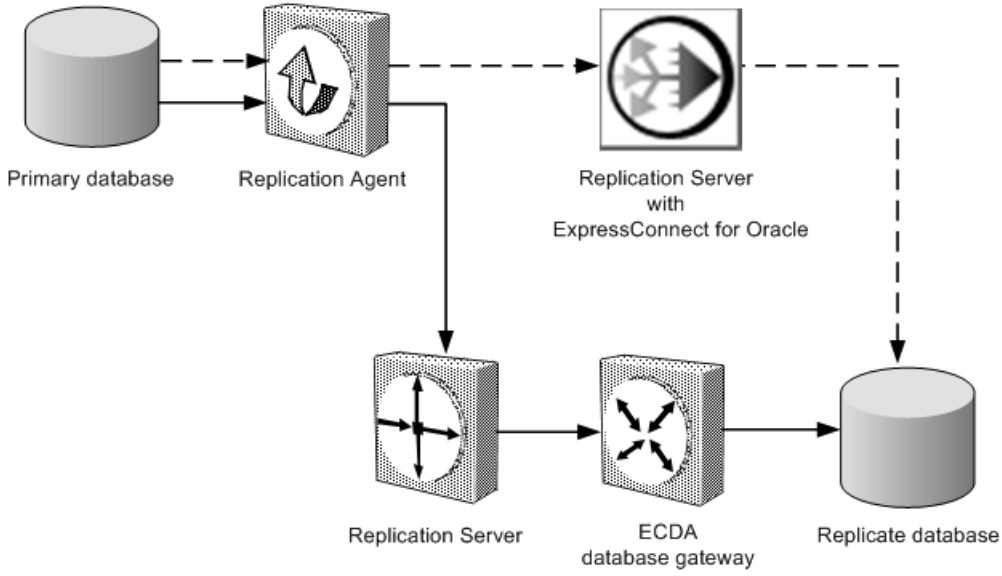
Additional components are required to implement a Sybase replication system with non-ASE data servers:

- A Replication Agent
- Enterprise Connect™ Data Access (ECDA) or a data server for which connectivity requirements are compatible with Replication Server, or ExpressConnect for Oracle.

Figure 1-2 illustrates a typical Sybase replication system with non-ASE data servers, showing the flow of data between the data servers, through:

- Replication Agent, Replication Server, and Enterprise Connect Data Access database gateway, and,
- Replication Agent, Replication Server, and ExpressConnect for Oracle.

Figure 1-2: Sybase replication system with non-ASE data servers



If you are using ECDA database gateways, data flows from the primary database to the Replication Agent, from the Replication Agent to the Replication Server, from the Replication Server to the DirectConnect database gateway, and then from the database gateway to the replicate database.

ECDA database gateways support IBM DB2 UDB, Microsoft SQL Server, and Oracle data servers by providing connectivity between Sybase Open Client and Open Server and either ODBC or the native protocol of the replicate data server, and by providing SQL transformation and other services. Replication Server also includes datatype support for non-ASE data servers.

If you are using ExpressConnect for Oracle, data flows from the primary database to the Replication Agent, from the Replication Agent to the Replication Server, and then from the Replication Server directly to the replicate database.

Replication Agents support non-ASE primary data servers by reading the completed transactions in the primary database and sending them to Replication Server for distribution.

Replication system components

The following components are described by their function and role in a Sybase replication system:

- Primary data server
- Replication Agent
- Replication Server
- Database gateway
- ExpressConnect for Oracle
- Replicate data server

For a more complete description of the Sybase software products (Replication Server, Replication Agents, and Enterprise Connect Data Access gateways), see Chapter 2, “Replication Components Detail.”

Primary data server

A **primary data server** manages one or more primary databases, which are the sources of the data-changing operations or transactions in a replication system. The primary data server is configured to capture information needed for replication.

All primary data servers are supported by Replication Agents. ASE has an internal Replication Agent. The non-ASE servers require an external Replication Agent.

Supported primary database servers

In addition to Adaptive Server Enterprise, Sybase replication technology actively supports transaction replication from the following relational database servers:

- IBM DB2 UDB on z/OS
- IBM DB2 UDB on UNIX/Windows
- Microsoft SQL Server
- Oracle

To find out about the most current, supported versions of these data servers, see the documentation for the Replication Agent that supports a particular non-ASE data server.

General issues for non-ASE primary data servers

There are several issues related to non-ASE primary data servers in a Sybase replication system. A successful replication system must address all of the issues for each primary data server.

Non-ASE primary data server issues include:

- The requirements of the Replication Agent, including any limitations, intrusions, and impacts on the data server's operation
- The access and permissions necessary for the Replication Agent to obtain transactions from the primary database
- The connectivity requirements to support communication between the primary data server and other replication system components
- The limitations imposed on the replication system by the non-ASE data server
- The datatype conversions (or translations) that may be required to replicate transactions from one type of data server to another
- The replication system management issues specific to the non-ASE data server

The following sections describe the specific primary data server issues for each actively supported type of non-ASE data server.

Replication Agent

A **Replication Agent** transfers transaction information, which represents changes made to data schemas and execution of stored procedures, from a primary data server to a Replication Server, for distribution to other (replicate) databases.

In Adaptive Server Enterprise, an embedded Replication Agent is provided with the database management system software. The Replication Agent for ASE is called Replication Agent, and it is an Adaptive Server thread.

For non-ASE data servers, Sybase provides Replication Agent products:

- Replication Agent for DB2 UDB – provides primary data server support for IBM DB2 UDB servers that run on IBM z/OS platforms.
- Replication Agent – provides primary data server support for DB2 UDB, Microsoft SQL Server, and Oracle data servers that run on Linux, UNIX, or Microsoft Windows platforms.

Replication Agents read the primary database transaction log. The primary Replication Server reconstructs the transaction and forwards it to replicate sites that have subscriptions for the data.

A Replication Agent is required for each database that contains primary data or for each database where replicated stored procedures are executed.

Replication Server

A **Replication Server** at each primary or replicate site coordinates data replication activities for local data servers and exchanges data with Replication Servers at other sites.

Replication Server performs the following major tasks:

- Receives transactions from primary databases through a Replication Agent and distributes them to replicate database sites that have subscriptions for the data

- Receives transactions from other Replication Servers and applies them to local replicate databases or forwards them to other replication servers that have subscriptions for the data
- Provides guaranteed delivery of transactions to each replicate site

The information needed to accomplish these tasks is stored in Replication Server system tables. The system tables include descriptions of the replicated data and replication objects, such as replication definitions and subscriptions, security records for Replication Server users, routing information for other Replication Server sites, access methods for local databases, and other administrative information.

Replication Server system tables are stored in a database called the *Replication Server System Database* (RSSD). Alternately, the Replicate Server can use an embedded RSSD (ERSSD). Each Replication Server has its own RSSD.

Database gateway

A **database gateway** allows clients using one communication protocol to connect with data servers that use a different protocol.

The Sybase Enterprise Connect Data Access product line consists of database gateway servers that allow clients using the Sybase Open Client and Open Server protocol (such as Replication Server) to connect with non-Sybase data servers, using either the data server's native communication protocol or the standard, ODBC protocol.

Sybase Enterprise Connect Data Access products also allow for the retrieval of metadata from non-ASE replicate data servers.

For more information about Enterprise Connect Data Access database gateways, see "ECDA" on page 45.

ExpressConnect for Oracle

ExpressConnect for Oracle, which is available with Replication Server Options 15.5 and later, provides direct communication between Replication Server and a replicate Oracle data server. ExpressConnect for Oracle eliminates the need for installing and setting up a separate gateway server, thereby improving performance and reducing the complexities of managing a replication system. See "ExpressConnect for Oracle" on page 49.

Replicate data server

A **replicate data server** manages a database that contains replicate data, which is data that is a "copy" of the data in a primary database.

Replication Server maintains the data in a replicate data server by logging in as a database user. In the case of non-ASE data servers, Replication Server logs in to the replicate data server through a database gateway server or directly to the data server.

Replication Server can treat any server as a data server if it supports a set of required data operations and transaction processing directives, either directly (such as Adaptive Server Enterprise) or indirectly (such as a Enterprise Connect Data Access database gateway server).

Supported replicate database servers

In addition to Adaptive Server Enterprise, Sybase replication technology supports transaction replication into the following relational database servers:

- IBM DB2 UDB on z/OS
- IBM DB2 UDB on UNIX/Windows
- Microsoft SQL Server
- Oracle
- Sybase IQ

For more information regarding the current supported versions of Oracle, Microsoft SQL Server, and DB2 UDB data servers, refer to the documentation for the ECDA database gateway associated with a particular non-ASE data server. For information on the supported Oracle data server version for ExpressConnect, see the *ExpressConnect for Oracle Installation and Configuration Guide*.

General issues for non-ASE replicate data servers

There are several issues related to non-ASE replicate data servers in a Sybase replication system. A successful replication system must address all of the issues for each replicate data server.

Non-ASE replicate data server issues include:

- The requirements of the ECDA database gateway for the non-ASE data server.

- The access and permissions required in the replicate data server for the replication system to apply transactions to the replicate database.
- The connectivity requirements for communication between the replicate data server and other components of the replication system.
- The limitations on replication into the non-ASE data server.
- The intrusion and impact of the database objects required to support Replication Server operations.
- The replication system management issues specific to the non-ASE data server.

Non-ASE replication issues

The biggest challenge in implementing a successful heterogeneous replication system is accommodating the unique characteristics of data servers that are supplied by different vendors. Regardless of the type or brand of a data server, there are issues that are specific to the data server's role in the replication system. When a single data server acts as both a primary data server and a replicate data server (bidirectional replication), there are still more issues to consider.

Primary database issues

The following primary database issues must be addressed in a successful heterogeneous replication system:

- The requirements of the Replication Agent and the intrusions and impacts of the Replication Agent on the data server. For example, some Replication Agents create and use database objects in the primary database to support replication.
- The access and permissions required in the data server for other replication system components. Both the primary Replication Server and the Replication Agent for a database must have user IDs and passwords defined in the database with appropriate permissions to access primary database objects.

- The connectivity required to support communication between the data server and other replication system components. Replication Agents use the native communication protocol of the data server, ODBC protocols, or JDBC protocols to communicate with the primary database. Replication Server may require a database gateway to communicate with a data server.
- The specific limitations on replication from the particular data server. For example, some Replication Agents restrict the configuration options of some data servers. Replication Server may impose size limitations on some native datatypes in some databases.
- How replication definitions stored in the RSSD are used by the Replication Agent for the particular data server. For example, both Replication Server and Replication Agents are case-sensitive in identifying database object names, but some databases are not.
- The datatype conversions that may be required when replicating transactions from one particular data server to another type of data server. For example, almost every type of data server has a unique way of representing temporal data. The TIMESTAMP datatype in one database may need to be "translated" to be stored as a datetime datatype in another database.
- The replication system management issues specific to the particular data server. For example, different data servers allow different system management options.

For more information about specific primary database issues for specific databases, see the appropriate chapter for your database.

Replicate database issues

The following replicate database issues must be addressed in a successful heterogeneous replication system:

- The requirements of the ECDA database gateway for the particular database server. Configure the DirectConnect access services to work with the replicate database server and Replication Server.
- The access and permissions required in the data server for the replication system to apply transactions to the replicate database. Both the replicate Replication Server and the ECDA gateway for a database must have user IDs and passwords defined in the database, with appropriate permissions to access replicate database objects.

- The connectivity required to support communication between the replicate data server and other replication system components. ECDA gateways use either the native communication protocol of a data server, or standard ODBC or JDBC protocols to communicate with a replicate database. Replication Server generally requires a database gateway to communicate with a non-ASE data server.
- The limitations on replication into the particular data server. For example, Replication Server imposes limitations on some native datatypes in some databases.
- The intrusion and impact of the database objects required to support Replication Server operations. Replication Server requires two tables and may require some stored procedures to manage a replicate database.
- The replication system management issues specific to the particular data server. For example, different data servers allow different system management options.

For more information about specific replicate database issues for specific databases, see the appropriate chapter for your database.

Setting character sets

In a heterogeneous replication system, in which the primary and replicate data servers are different types, servers may not support all the same character sets. In such cases, replication system components must perform at least one character set conversion (from the primary data server's character set to the replicate data server's character set).

Even in a homogeneous replication system, in which both primary and replicate data servers are the same type, character set conversions might be required if replication system components reside on more than one type of platform.

Character set problems can produce data inconsistencies between the primary database and the replicate database. To avoid character set problems, you must either:

- Use the same character set on all servers and platforms in the replication system, or
- Use compatible character sets on all servers and platforms in the replication system, and configure replication system components to perform the appropriate character set conversions.

For more information about setting and overriding the default character set, see the appropriate Replication Agent documentation.

Heterogeneous replication limitations

There are some possible limitations of a heterogeneous replication system depending on the particular databases involved, and based on Sybase replication technology:

- Stored procedure replication
- Owner-qualified object names
- Large object replication
- Setup for replicate databases
- Replication Server support for encrypted columns
- Subscription materialization
- Replication Server rs_dump command
- Replication Server rs_marker command
- Replication Server rs_dumptran command
- Replication Server rs_subcmp utility
- Dynamic SQL
- Bulk copy
- Replication Server rs_ticket stored procedure

Stored procedure replication

Stored procedure replication allows the execution call of a stored procedure to be replicated, including the parameter values passed as arguments to the primary stored procedure call.

The availability of stored procedure replication depends on the capabilities of the primary and replicate databases, as well as support from the associated Replication Agent and ECDA database gateway. Refer to the documentation for the specific Replication Agent and ECDA components to determine if stored procedure replication is available for your databases.

Owner-qualified object names

Access to replicate tables and stored procedures in a non-ASE database often requires that the reference to the replicate table or stored procedure be owner-qualified.

For example, suppose the Replication Server maintenance user assigned to apply transactions to an Oracle replicate database is orauser. A replicate insert command to table table1 may fail with a "table not found" error if the owner of table1 is bob. When attempting to find table1, Oracle looks for orauser.table1, not bob.table1. To properly identify the replicate table to be updated, you can:

- Create an alias at the Oracle replicate database that refers to the correct replicate table. For example, create a synonym object in Oracle named table1, which refers to the fully qualified name of "bob.table1."
- When creating the replication definition, use the with replicate table named [table_owner.['table_name']] clause. Continuing with the same example, the clause is:

```
with replicate table named bob.table1
```

Owner qualifying with multiple replicate databases

The problem becomes a little more complicated when table1 is to be replicated to more than one replicate database (for example, Oracle replicate table bob.table1). The option of using the with replicate table named clause in the replication definition supports only one replicate table name.

To work around this issue, create multiple replication definitions, one for each unique replicate table name required. Make sure each subscription refers to the correct replication definition and each replication definition uses the with replicate table named clause.

Large object replication

Large object (LOB) datatypes (such as BLOB, CLOB, IMAGE, and TEXT) provide support for the longest streams of character and binary data in a single column. The size of the LOB datatypes poses unique challenges, both as primary and replicate data.

Primary database LOB replication issues

At the primary database, the impact of LOB datatypes is on the transaction logging function. For Replication Agents, the log resources must be adequate to support retention of the changes in LOB data, only after images of LOB data are logged. The ability of LOB replication depends on the capabilities of the Replication Agent.

Replicate database LOB replication issues

Adaptive Server Enterprise uses a text pointer to identify the location of text and image column data. The text pointer is passed to system functions that perform the actual updates to data in these large columns. The same technique is used internally in Replication Server to apply LOB datatypes. Replication Server obtains a text pointer, and data server function calls are made to apply the data to replicate databases.

When a non-Sybase database is the replicate database, the database gateway used to communicate with the replicate database must be able to emulate the Adaptive Server text pointer processing. The ECDA Option for ODBC, ECDA Option for Oracle, ExpressConnect for Oracle, and the Mainframe Connect™ DirectConnect for z/OS Option gateways provide this feature.

ECDA Option for ODBC

The ECDA Option for ODBC provides support for LOB replication into Microsoft SQL Server databases. See Chapter 9, “Microsoft SQL Server Replicate Data Server Issues.”

Setup for replicate databases

Replication Server provides a utility named rs_init, which sets up an Adaptive Server database as a primary or replicate database as follows:

- Creates the Replication Server database connection
- Creates the required tables and stored procedures in the replicate database
- Defines the Replication Server maintenance user ID

Heterogeneous replication support does not include a utility that is equivalent to rs_init. Instead, Replication Server commands for creating connections, and primary and replicate data server commands for creating objects that did support replication including a maintenance user, may be used. In Replication Server 15.2, the introduction of the "using profile" clause of the create connection command may be used to accomplish many of these tasks.

Replication Server support for encrypted columns

Replication Server supports replication of encrypted column data between Adaptive Server databases. However, replication of encrypted column data to any non-ASE replicate database is not supported.

To replicate non-encrypted data to an ASE database containing an encrypted column, disable the rs_set_ciphertext function string for the Adaptive Server connection. The rs_set_ciphertext function string is executed for all ASE connections by default. It indicates to the replicate ASE database that the data to be replicated is already encrypted and the assumption is that the primary database is also an ASE with the same encryption usage. By disabling the rs_set_ciphertext function string, you allow the replicate ASE to perform encryption on the incoming replicated data. Allowing ASE to encrypt the incoming data is appropriate if the primary database is non-ASE, or if the primary ASE database does not use enrypted columns.

Function string

rs_set_ciphertext

Description

Controls replication of encrypted columns to an Adaptive Server table.

Example

Alter function string rs_set_ciphertext to turn off execution of the ASE-specific command "set ciphertext on."

```
alter function string rs_set_ciphertext
for some_function_string_class
output language
''
```

Subscription materialization

Materialization is 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 Replication Server:

- Bulk materialization – 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 – creating a subscription and populating a replicate database using Replication Server commands.

Heterogeneous replication supports bulk materialization methods with varying complexity based on the specific Replication Agent capabilities.

See the *Replication Server Administration Guide* for a general discussion of subscription materialization, and see the appropriate Replication Agent documentation for details regarding a particular Replication Agent and its materialization support.

Replication Server *rs_dump* command

The Replication Server rs_dump command is typically used to coordinate database dump activities across a replication system. When a replicate connection receives an rs_dump transaction, Replication Server executes the rs_dump function string for that connection. You can customize the rs_dump function string to execute whatever commands are required.

For non-ASE primary database replication, some Replication Agents provide a method to invoke the rs_dump command from a non-Sybase primary database. Refer to the appropriate Replication Agent documentation to determine if rs_dump execution from the primary database is supported.

For replicate databases, no default function string for rs_dump is provided.

For more information about the rs_dump command, its use, and function-string modifications, see the *Replication Server Reference Manual*.

Replication Server *rs_marker* command

The Replication Server rs_marker command is a primary database transaction log marker mechanism, which assists with the materialization process. An rs_marker execution passes activate subscription and validate subscription commands to a primary Replication Server. Most Replication Agents support an rs_marker invocation to assist with materialization.

For more information about rs_marker usage, see the *Replication Server Reference Manual*. For more information about the use and availability of rs_marker for a particular database, see the appropriate Replication Agent documentation.

Replication Server *rs_dumptran* command

The Replication Server rs_dumptran command is typically used to coordinate database transaction dump activities across a replication system. When a replicate connection receives an rs_dumptran transaction, the Replication Server executes the rs_dumptran function string for that connection. You can customize the rs_dumptran function string to execute whatever commands are required.

Heterogeneous replication does not support rs_dumptran for non-Sybase primary databases.

For replicate databases, no default function string for rs_dumptran is provided.

For more information about the rs_dumptran command, its use, and function-string modifications, see the *Replication Server Reference Manual*.

Replication Server *rs_subcmp* utility

Replication Server provides an rs_subcmp executable program that you can use to compare primary and replicate tables, optionally reconciling any differences found.

For non-Sybase database support, you may use rs_subcmp, providing you have connectivity to the primary and replicate databases. You must also develop custom SELECT commands for the primary and replicate databases to generate comparable outputs for both. Additional options are to buy third-party tools that provide such functionality, or build your own application.

For more information about comparing and reconciling databases in a heterogeneous replication system, see Appendix C, “Heterogeneous Database Reconciliation.”

Dynamic SQL

Dynamic SQL allows the Replication Server Data Server Interface (DSI) to prepare dynamic SQL statements at the target user database and to run them repeatedly.

Dynamic SQL is available for Oracle, DB2 UDB z/OS, and DB2 UDB on UNIX, Windows, and Linux. It is not available for Microsoft SQL and Sybase IQ.

Bulk copy

Bulk-copy allows Replication Server Data Server Interface (DSI) to improve performance when replicating large batches of insert statements on the same table using the Open Client Open Server Bulk-Library interface.

Bulk-copy is not available for any of the non-ASE data servers, with the exception of Oracle, when using ExpressConnect for Oracle.

Replication Server *rs_ticket* stored procedure

rs_ticket is a stored procedure in the primary database that you can use to help monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce.

rs_ticket is available for Oracle, DB2 UDB on UNIX, Windows, and Linux, and Microsoft SQL. It is not available for DB2 UDB z/OS.

See the *Replication Agent Reference Manual*.

Replication system non-ASE configurations

This section discusses several replication system configurations with heterogeneous or non-ASE data servers, and describes the issues with each configuration.

Non-ASE primary to Adaptive Server replicate

The simplest heterogeneous replication scenario is replicating one-way from a non-ASE primary database to an Adaptive Server replicate database. The only unique requirements are a Replication Agent designed to extract transaction data from the non-ASE primary database, and the application of the Heterogeneous Datatype Support (HDS) feature of Replication Server to translate primary database native datatypes to Adaptive Server datatypes. See "Translating Datatypes using HDS" in the *Replication Server Administration Guide*.

Replication system components

The following components are required for a non-ASE primary to Adaptive Server replicate configuration:

- Non-ASE primary data server. For example, Oracle.
- Replication Agent designed for the primary data server
- Replication Server
- Adaptive Server replicate data server

Replication system issues

In a non-ASE primary to Adaptive Server replicate configuration, the Replication Server database connection for the primary database may require a valid user ID and password for the primary database (validated only for Replication Agent), even though this user ID does not apply transactions to the primary database.

ASE server primary to non-ASE server replicate

A simple heterogeneous replication scenario is replicating one-way from an Adaptive Server primary database to a non-ASE replicate. The only unique requirements are an ECDA database gateway to apply transaction data to the replicate database, and the application of the HDS feature of Replication Server to translate Adaptive Server datatypes to the native datatypes of the replicate database. For more detailed information about HDS, see "Translating Datatypes using HDS" in the *Replication Server Administration Guide.*

Replication system components

The following components are required for an Adaptive Server primary to non-ASE replicate configuration:

- Adaptive Server primary database
- Replication Server
- ECDA database gateway designed for the replicate data server. For example, ECDA Option for ODBC (for Microsoft SQL Server).
- Non-ASE replicate data server. For example, Microsoft SQL Server.

Replication system issues

Consider the following issues in an Adaptive Server primary to non-ASE replicate configuration:

- The Replication Server database connection for the replicate database must include a valid user ID and password (the maintenance user) for the replicate database. This user ID must have authority to apply replicate transactions in the replicate database.
- The Replication Server replicate database connection must be created using the correct profile for the replicate database. The connection profile specifies the correct function-string class and error class for the replicate database, and additionally may contain class-level translation definitions and replicate database object creation, to support replication.

Non-ASE primary to non-ASE replicate

This scenario varies in complexity, depending on the mix of non-ASE data servers.

Replication system components

The following components are required for a non-ASE primary to non-ASE replicate configuration:

- Non-ASE primary data server. For example, Oracle.
- Replication Agent designed for the primary data server. For example, Replication Agent for Oracle.
- Replication Server
- Gateway designed for the replicate data server (for example ECDA Option for ODBC, ExpressConnect for Oracle).
- Non-ASE replicate data server. For example, Microsoft SQL Server.

Replication system issues

Consider the following issues in a non-ASE primary to non-ASE replicate configuration:

- The Replication Server primary database connection may require a valid user ID and password for the primary database. This user ID must have authority to apply replicate transactions (even if no transactions will be replicated to the primary database).
- The Replication Server replicate database connection must be created using the correct profile for the replicate database. The connection profile specifies the correct function string classes and error classes for the replicate database, and additionally may contain class-level translation definitions and replicate database object creation, to support replication.

Bidirectional non-ASE to non-ASE replication

In this scenario, replication occurs both to and from each database. Each non-ASE database must have both a Replication Agent *and* an ECDA database gateway.

Replication system components

The following components are required for a bidirectional non-ASE primary to non-ASE replicate configuration:

- Non-ASE primary data server. For example, DB2 UDB on UNIX, Windows, and Linux.
- Replication Agent designed for the primary data server. For example, Replication Agent for Oracle, Microsoft SQL Server, and DB2 UDB.
- ECDA database gateway designed for the “primary” data server acting as a replicate database. For example, ECDA Option for ODBC (for DB2 UDB).
- Replication Server
- ECDA database gateway designed for the replicate data server. For example, ECDA Option for ODBC (for Microsoft SQL Server).
- Replication Agent designed for the “replicate” data server acting as a primary database. For example, Replication Agent for Linux, Microsoft Windows, and UNIX.
- Non-ASE replicate data server. For example, Microsoft SQL Server.

Replication system issues

From a technical standpoint, you can set up a bidirectional replication scenario using only two Replication Server database connections (one “primary-and-replicate” connection for each database).

Note In the following description of bidirectional replication issues, the two databases are referred to as Database #1 and Database #2, because both databases take on both “primary” and “replicate” roles in the replication system.

Consider the following issues in a bidirectional non-ASE primary to non-ASE replicate configuration:

- The Replication Server primary database connection for Database #1 must include a valid user ID and password for the primary database. This user ID must be the same user ID specified in the Replication Server replicate database connection for Database #2 (the maintenance user). This user ID must have authority to apply transaction operations to replicate tables in Database #1.

- The Replication Agent for Database #1 must be configured to bypass maintenance user transactions to prevent a transaction from returning from the replicate tables in Database #2. See the appropriate Replication Agent documentation for details on configuring the Replication Agent to bypass maintenance user transactions.
- The Replication Server primary database connection for Database #2 must include a valid user ID and password for the primary database. This user ID must be the same user ID specified in the Replication Server replicate database connection for Database #1 (the maintenance user). This user ID must have authority to apply transaction operations to replicate tables in Database #2.
- The Replication Agent for Database #2 must be configured to bypass maintenance user transactions to prevent a transaction from returning from the replicate tables in Database #1. Refer to the appropriate Replication Agent documentation for details on configuring the Replication Agent to bypass maintenance user transactions.
- The Replication Server replicate database connections to Database #1 and Database #2 must be created using the correct profile for the replicate database. The connection profile specifies the correct function-string classes and error classes for the replicate database, and additionally may contain class-level translation definitions and replicate database object creation, to support replication.

CHAPTER 2

Replication Components Detail

This chapter describes in greater detail the Sybase software products that allow you to implement a Sybase replication system with heterogeneous or non-ASE data servers.

Sybase replication products

Sybase offers product lines that specifically support replication systems with heterogeneous or non-ASE data servers, based on Sybase replication technology:

- Replication Server, which is the centerpiece of Sybase advanced replication technology and incorporates several features specifically to support non-ASE data servers in a Sybase replication system.
- Replication Server Options that consist of a Replication Agent and either an Enterprise Connect Data Access (ECDA), or ExpressConnect for Oracle.
    - Replication Agents support Replication Server by providing a way to obtain replication data from non-ASE primary databases. Replication Agents provide this support for DB2 UDB, Microsoft SQL Server, and Oracle data servers.
    - ECDA database gateways support Replication Server by providing access to a variety of non-ASE databases, allowing them to function as replicate databases in a Sybase replication system.

    - ExpressConnect for Oracle supports Replication Server by providing direct communication between Replication Server and Oracle database, without a need for a separate gateway server. Express Connect for Oracle is only available with Replication Server Options 15.5 or later.
- Replication Agent for IBM DB2 UDB that replicates data from IBM DB2 UDB on the mainframe.

Replication Server

Replication Server can access data locally instead of from remote, centralized databases. Compared to a centralized data system, a replication system improves system performance and data availability, and reduces communication overhead. Replication Server provides a cost-effective, fault-tolerant system for replicating data.

Because Replication Server replicates transactions—incremental changes instead of data copies—and stored procedure invocations, rather than the operations that result from execution of the stored procedures, it enables a high-performance distributed data environment while maintaining transactional integrity of replicated data across the system.

How Replication Server works

Replication Server works to distribute data over a network by:

- Providing application developers and system administrators with a flexible publish-and-subscribe model for marking data and stored procedures to be replicated
- Managing replicated transactions while retaining transaction integrity across the network

A Replication Server at each primary or replicate site coordinates the data replication activities for the local data servers and exchanges data with Replication Servers at other sites.

A Replication Server:

- Receives transactions from primary databases through Replication Agents and distributes them to sites with subscriptions for the data

- Receives transactions from other Replication Servers and applies them to local databases

Replication Server system tables store the information needed to accomplish these tasks. The system tables include descriptions of the replicated data and the following replication objects:

- Replication definitions and subscriptions
- Security records for Replication Server users
- Routing information for other sites
- Access methods for local databases
- Other administrative information

Replication Server system tables are stored in a database called the Replication Server System Database (RSSD).

To manage replication information in Replication Server, use Replication Command Language (RCL). You can execute RCL commands, which resemble SQL commands, on Replication Server using isql, the Sybase interactive SQL utility. For a complete reference for RCL, see the *Replication Server Reference Manual*.

Publish-and-subscribe model

Transactions that occur in a primary database are detected by a Replication Agent and transferred to the local Replication Server, which distributes the information across a network to Replication Servers at destination sites. In turn, these Replication Servers update the replicate database according to the requirements of the remote client.

The primary data is the source of the data that Replication Server replicates in other databases. You publish data at primary sites to which Replication Servers at other (replicate) sites subscribe. To do so, you first create a *replication definition* to designate the scope and location of the primary data. The replication definition describes the structure of the table. A database replication definition can replicate individual tables, functions, and DDLs. A table replication definition describes the structure of the table and states the key that is to be used to query the table for updates and deletes.

Creating a replication definition does not, by itself, cause Replication Server to replicate data. You must also create a *subscription* against the replication definition to instruct Replication Server to replicate the data in another database. A subscription resembles a SQL select statement: It can include a where clause to specify the rows of a table you want to replicate in the local database.

You can have multiple replication definitions for a primary table to filter different objects. Replicate tables can subscribe to different replication definitions to obtain different views of the data.

After you have created subscriptions to replication definitions or publications, Replication Server replicates transactions to databases with subscriptions for the data.

Replicated functions

With some data servers, Replication Server allows you to replicate stored procedure invocations asynchronously between databases. By encapsulating many changes in a single replicated function, you can improve performance over normal data replication. Because they are not associated with table replication definitions, replicated functions can execute stored procedures that may or may not modify data directly.

Note Replication Server does not support stored procedure replication on all types of data servers. For more information about replicating stored procedures on a particular data server, refer to the appropriate Replication Agent documentation.

With replicated functions, you can execute a stored procedure in another database. A replicated function allows you to:

- Replicate the execution of a stored procedure to subscribing sites
- Improve performance by replicating only the name and parameters of the stored procedure rather than the actual database changes

Replication Server supports both *applied functions* and *request functions*:

- An *applied function* is replicated from a primary to a replicate database. Create subscriptions at replicate sites for the function replication definition and mark the stored procedure for replication in the primary database.

- A *request function* is replicated from a replicate to a primary database. There is no subscription for a request function. Mark the stored procedure for replication in the replicate database.

Transaction management

Replication Server depends on data servers to provide the transaction processing services needed to protect stored data. To guarantee the integrity of distributed data, data servers must comply with such transaction-processing conventions as atomicity and consistency.

Data servers that store primary data provide most of the concurrency control needed for the distributed database system. If a transaction fails to update a table with primary data, Replication Server does not distribute the transaction to other sites. When a transaction does update primary data, Replication Server distributes the changes, and unless a failure occurs, the update succeeds at all sites that have subscribed to the data.

Relationship with other system components

Replication Server interacts with other components of a replication system as either a *server* or a *client.*

As a server, Replication Server supports connections from:

- Replication Agents, across which database commands are sent from primary databases
- Other Replication Servers, thus distributing the processing involved in message delivery and providing a degree of scalability in a replication system
- Users or management tools for administration, data server identification, message publication and subscription, and so on

As a client, Replication Server connects to:

- A Replication Server System Database (RSSD) which can be on an external Adaptive Server Enterprise database, or the internal embedded RSSD (ERSSD).
- A database gateway to connect to the replicate non-ASE database.

- The Oracle replicate database directly, when using ExpressConnect for Oracle.

Replication Server communication protocols

Replication Server is an Open Client and Open Server application that uses Sybase Tabular Data Stream™ (TDS) as the underlying communication protocol. Any clients that request services from Replication Server must implement an Open Client interface. This includes Replication Agents, system management tools, and user interface tools such as isql.

As a client distributing messages to other Replication Servers or to replicate data servers, Replication Server uses an Open Client interface. Therefore, when Replication Server needs to send a message to a data server, either that data server must support an Open Server interface running on TDS, or there must be an Open Server/TDS bridge or gateway application between Replication Server and the replicate data server.

Replication to Sybase IQ does not require an additional gateway software because it appears as an Open Server to Replication Server. Except for Oracle, which can also use ExpressConnect to directly connect to the replicate database, the gateway software for replicating to DB2 UDB, Microsoft SQL Server, and Oracle, is in the form of a Sybase ECDA database gateway. Some ECDA gateways bridge from Open Server/TDS to the native interface of the replicate data server (for example, ECDA Option for Oracle), while others bridge from Open Server/TDS to an ODBC or JDBC driver for the data server. Replication Server configurations vary, depending on the gateway used.

Replication to Oracle using ExpressConnect does not require an additional gateway; ExpressConnect uses native Oracle connectivity, allowing Replication Server using ExpressConnect to connect directly to Oracle.

Replication Server user IDs and permissions

Replication Server requires several different user IDs. Some user IDs are required for other components (or users) to access the Replication Server, and others are required for the Replication Server to have access to other components in a replication system.

Define user IDs in the Replication Server using the Replication Server create connection command.

Note Depending on how your replication system is configured, some of the user IDs in the following list might not be required. For example, if you have separate Replication Servers for primary and replicate databases, the primary Replication Server does not require a user ID to access a replicate database.

The following user IDs are defined in a Replication Server:

- Replication Agent user – used by a Replication Agent to log in to a primary Replication Server. This user ID must have connect source permission to deliver database commands through the LTL interface.
- Replication Server user – used by other Replication Servers to log in to a Replication Server and forward messages. This user ID must have connect source permission to forward database commands through the RCL interface.
- SysAdmin user – used by system administrators or system administration tools to perform administration activities. Depending on the task, this user ID must have sa, create object, or primary subscribe permission.
- Maintenance user – used by Replication Server to deliver messages to a replicate data server. This user ID must have the necessary permissions in the replicate data server to execute the commands to which messages to be delivered are mapped to a primary database. Work performed by the maintenance user is not replicated.
- Replicate user – used by a replicate Replication Server to deliver messages to a primary data server. For delivery for "request" messages, that is, messages from a replicate data server that are selected for delivery to the primary data server, Replication Server uses the user ID of the user who executes the command in the replicate database. This user ID must have the necessary permissions in the primary data server to execute the commands to which messages to be delivered are mapped.
- RSI user – used by Replication Server to log in to other Replication Servers to forward messages to be delivered. This user ID must have connect source permission in the replicate Replication Server.
- RSSD user – used by Replication Server to log in to the Replication Server System Database (RSSD) that manages its operational data. This user ID must have full control in the RSSD to create and drop objects, execute procedures, and query and update tables.

Relationship with Replication Agents

While Replication Server is extensible (customizable function strings and error handling, custom datatype definitions, and translations between datatypes) to meet the needs of replicate data servers, Replication Server support of primary data servers is limited.

The Replication Server interface for primary data servers is its proprietary Log Transfer Language (LTL). Transactions from a primary data server must be translated to LTL to be delivered to a primary Replication Server. Therefore, support for primary data servers is limited to those for which Sybase provides a Replication Agent to perform the translation to LTL for primary database operations.

Replication Server interfaces on both the primary and replicate sides are supported by the underlying Open Client/Open Server interface running on TDS.

LTM locator updates

The primary Replication Server maintains a "locator" value (LTM locator) that identifies the last point in a transaction log from which all data has been successfully received by the primary Replication Server. The Replication Agent periodically requests this value from the Replication Server connection to identify a position in the transaction log, which can then be used to identify where older data can be released or removed from the log.

There is a performance trade-off in determining how often to request an LTM locator update. Frequent queries of the LTM locator value from a Replication Server can slow down replication (the Replication Agent must stop sending LTL commands long enough to request and receive the LTM locator value) while it provides more frequent opportunities to release data from the primary database transaction log. When restarting, the Replication Agent must re-send all data in the log that exists since the last LTM locator value was received from Replication Server.

Generally, if replication throughput performance is a priority, acquire enough log resource to allow less frequent log truncation and less frequent retrieval of the LTM locator value. If log resources are scarce, more frequent retrieval of the LTM locator value and more frequent truncation may be necessary.

For more information about using the LTM locator, see the appropriate Replication Agent documentation.

LTL generation

The number of bytes of information sent to Replication Server has a direct impact on the performance of the replication system; more data and commands received by Replication Server require more work and time to process. In addition, more data also requires more network resources. There are several configuration options available for the Replication Agent that you can use to minimize this impact:

- Using the RSSD. By reading replication definitions from the RSSD, the Replication Agent can send the column data in the same column order as specified by the replication definition. This allows Replication Server to bypass sorting the column information before processing. Furthermore, column names are not sent with the data, which reduces the number of bytes of information required.
- Sending minimal columns. When an update operation occurs on a table, only a portion of the columns may have been altered. By sending the before and after images of only those columns that changed, the Replication Agent sends less information.

  Note Do not use minimal columns if the data in the replicate database involves custom function strings.

- Batch mode. A Replication Agent must "wrap" transactions in a limited amount of administrative LTL for the Replication Server. In batch mode, the Replication Agent can wrap multiple commands in the same set of administrative commands, which reduces the overall LTL generated and processed by the network and the Replication Server.

  In addition to batch mode, most Replication Agents have a "batch timeout" parameter, which allows a partial batch to be sent to the Replication Server after the Replication Agent waits a specified period of time and no additional transactions are received to fill the batch.

  Note Do not use Replication Agent batch mode if you use any Replication Server user-defined datatype (UDD) translations, either column-level or class-level.

- Origin time. Each transaction sent to Replication Server has an *origin queue ID*. The origin queue ID may include the time that the transaction was committed at the primary database. If the origin time is not sent by the Replication Agent, the processing effort is reduced somewhat, but the quantity of LTL sent to the Replication Server is the same.

For a complete description of the Replication Agent configuration parameters that affect LTL output, see the appropriate *Replication Agent Administration Guide*.

rs_ticket

Some Replication Agents can start rs_ticket transactions. These transactions provide data for Replication Server performance, module heartbeat, replication health, and table-level quiesce. See the *Replication Server Reference Manual.*

Database connections

Replication Server keeps track of other components in a replication system using *connections* that identify primary and replicate databases and *routes* that identify other Replication Servers.

Since Replication Server was originally designed for Adaptive Server Enterprise database replication, the definition of a connection in Replication Server follows the Sybase standard of <*server name*>.<*database name*>. For example, a Replication Server connection to an Adaptive Server named ASE1 and database PUBS is named ASE1.PUBS.

To connect to a primary non-ASE data server, Replication Server allows a connection from a Replication Agent on behalf of the non-ASE primary database. For a replicate database, Replication Server connects to an ECDA database gateway, which in turn connects to the non-ASE replicate data server. For Oracle, Replication Server can also connect directly to the replicate data server using ExpressConnect. Since Replication Agents, ECDA gateways, and ExpressConnect are not data servers, the Replication Server connection properties for those components may have different meanings than they do for a database server connection.

A single Replication Server connection can support data flow in either one or two directions. Data flows *in* through a Replication Server connection by way of the Replication Agent user thread. Data flows *out* through a Replication Server connection by way of the Data Server Interface (DSI) thread. Each Replication Server connection can support either outbound data flow only (through the DSI thread), or both inbound and outbound data flow (through the Replication Agent User and DSI threads).

Replication Agent User thread

Replication Server receives all data-change operations or transactions to be replicated from a primary data server through the Replication Agent User thread of the database connection for that data server. Every primary database that supplies transactions to be replicated must be represented by a Replication Server database connection with an enabled Replication Agent User thread.

Replication Server establishes a connection directly with the primary database, if it resides in an Adaptive Server. If the primary database resides in a non-ASE data server, a separate Replication Agent component communicates with the Replication Server, using a Replication Agent User thread connection, on behalf of the primary database.

Note Replication Server never attempts to connect to the Replication Agent User thread of a connection. The only entity that can initiate communication to a Replication Agent User thread is the primary data server or the Replication Agent.

On a Replication Agent User thread, the primary data server or Replication Agent is the client, and the primary Replication Server is the server.

DSI thread

The DSI thread of a Replication Server connection is where the replicated transaction is delivered by Replication Server. Every replicate database expected to receive replicated transactions must be represented by a Replication Server connection with an enabled DSI thread.

Replication Server establishes a connection directly with the replicate database, if it resides in Adaptive Server. If the replicate database resides in a non-Sybase data server, Replication Server communicates using:

- An ECDA database gateway by way of the connection's DSI thread, or,
- ExpressConnect to establish a connection directly with the Oracle replicate database.

Note A replicate data server or database gateway never attempts to connect to the DSI thread of a connection. The only entity that can initiate communication to a DSI thread is the Replication Server.

On a DSI thread, the Replication Server is the client, and the replicate data server or database gateway is the server.

Maintenance user purpose

To update replicated data, Replication Server logs in to the replicate data server as the maintenance user. The database owner (or the system administrator) must grant to the maintenance user the permissions required to insert, delete, and update rows in replicated tables, and to execute replicated stored procedures. In an Adaptive Server replicate database, Sybase Central or rs_init automatically creates the user ID for the Replication Server maintenance user and adds the user to the replicate database.

The maintenance user ID and password are defined to Replication Server automatically with the Replication Server create connection command for the replicate database. If you change the password for the maintenance user ID in the data server, you can use Sybase Central or the Replication Server alter connection command to change the password for the Replication Server connection.

The Replication Server maintenance user must also have permission to access the rs_lastcommit and rs_info system tables in the replicate database, and any stored procedures that use those tables.

Neither Sybase Central nor rs_init grants database permissions to the maintenance user for user tables and stored procedures. You must grant database permissions on replicated tables and stored procedures before you can replicate transactions for replicated tables or replicate executions of the replicated stored procedures. For each table replicated in the database, and for each stored procedure executed due to replication run:

```
grant all on table_name to maint_user
```

Alternatively, you can assign the maintenance user ID (maint_user) to a database administrator role, if that role has the required authority on all replicate objects.

DDL user purpose

Replication for Microsoft SQL Server and Oracle can replicate DDL commands that are entered at the primary database to the subscribers database. This capability is supported only where the primary and replicate data servers are identical, for example Oracle to Oracle. For more information, see the *Replication Agent Administration Guide*.

Datatypes and datatype definitions

Datatype definitions for a particular data server datatype are grouped in a *datatype class*.

For more information about datatype definitions (user-defined datatypes), see the description of the RSSD rs_datatype table in the *Replication Server Reference Manual*.

Restricted datatype

You cannot use the rs_address datatype as either the source or target of column-level or class-level translations.

Error and function-string classes for non-ASE data servers

Sybase provides function-string classes and associated function strings for all supported non-ASE replicate data servers. Non-ASE error classes are created by Replication Server and error actions are defined for different non-ASE error classes. You can create a connection to a non-ASE database with a corresponding error class by using the appropriate connection profile.

Object publication and subscriptions

The following limitations apply to object publications and subscriptions in a Sybase replication system:

- When declaring columns in a replication definition for a non-ASE primary database, use the Replication Server datatype that matches the datatype of the column in the primary database. If there is no matching native Replication Server datatype, find a datatype definition that matches the primary database datatype.
- When creating subscriptions with where clauses predicated on a column involved in column-level translation, specify the predicate value in "declared" format (that is, before translation).

Replication Agent

Replication Agent extends the capabilities of Replication Server by supporting non-ASE data servers as primary data servers in a Sybase replication system.

How Replication Agent works

A Replication Agent is a Replication Server client that retrieves information from a primary database transaction log and formats it for the primary Replication Server.

Begin by marking for replication the desired primary tables and stored procedures in the Replication Agent,.The Replication Agent detects any changes to primary data and using Log Transfer Language (LTL), a subset of Replication Control Language (RCL), sends primary data changes to the primary Replication Server.

A Replication Agent:

1 Logs in to the Replication Server.

2 Sends a connect source command to identify the session as a log transfer source and to specify the database for which transaction information will be transferred.

3 Retrieves the name of the maintenance user for the database from the Replication Server.

4 Requests the secondary truncation point for the database from the Replication Server.

5 Retrieves records from the transaction log, beginning at the record following the secondary truncation point, and formats the information into LTL commands.

Replication Agent connections

A Replication Agent sends data to Replication Server. Replication Agent logs in to the Replication Server, connects to the Replication Agent User thread of a Replication Server connection, and communicates with Replication Server over that connection. This has the following implications:

- A valid user ID, which the Replication Agent uses to log in to the Replication Server, must be defined at the Replication Server.
- The Replication Agent user ID must be granted connect source permission in Replication Server. connect source permission allows the Replication Agent to send commands that are valid only on a Replication Agent User thread.
- The Replication Agent must record this user ID and associated password.
- The Replication Agent must record the *server* and *database* portions of the Replication Server connection definition to identify and connect to the correct Replication Agent User thread.
- The user_name and password defined in the Replication Server create connection command may be a valid user ID and password for the primary database.

Note Replication Agent for Oracle, Microsoft SQL Server and IBM DB2 UDB for UNIX, requires the user_name and password to be valid and reports an error if the user is not found in the primary database.

The Replication Agent validates that the connection user_name exists in the primary database. However, Replication Server does not know if (or when) a DSI thread will be used. Therefore, the user ID and password must be valid in case the DSI thread is active.

Note The requirement for a valid primary database user ID varies by Replication Agent. Some Replication Agents do not require (nor do they check for) a valid user ID on the Replication Server connection.

Interfaces file

For the interaction between a Replication Agent and a Replication Server, the only *interface* file entry that may be required is one that identifies the Replication Server.

The Replication Agent for DB2 UDB does not require an *interface* file. The Replication Server and RSSD location, if needed, is in the *LTMCFG* file.

The Replication Agent (for DB2 UDB on UNIX and Windows platforms, Microsoft SQL Server, and Oracle) does not require an *interface* file entry, as it records the Replication Server host name and port number in configuration parameters.

Replication Agent maintenance user processing

When the Replication Agent connects to a Replication Server connection, the Replication Agent requests the maintenance user ID and may validate that the user ID exists in the primary database. This validation requires that the maintenance user ID defined in any Replication Server connection be valid for the database the connection represents, regardless of whether that connection is for primary transactions only, replicate transactions only, or both.

The Replication Agent does not use the maintenance user ID to log in to the primary database. Other than validating that the user ID exists, the only reference the Replication Agent makes to the maintenance user ID is to filter out primary database transactions created by the maintenance user.

The Replication Agent filters out maintenance user transactions to avoid having a transaction applied more than once to the primary database. In a bidirectional replication scheme, replication can occur both to and from the same database (which may have both a primary and a replicate role). When a primary transaction is applied to a replicate database, the applying user ID is the maintenance user for the replicate database. A Replication Agent scanning transactions at the replicate database must ignore the transactions applied by the Replication Server maintenance user to prevent those transactions from being sent back and applied to the primary database.

The Replication Agent accesses the database using a user ID defined at the primary database (or for DB2, a user ID that can access the DB2 log files). This user ID is not the same as the maintenance user defined in the Replication Server connection. The Replication Agent user ID used to access the primary database has a different role and purpose than the maintenance user defined to apply replicated transactions.

There may also be another user ID defined to the Replication Agent that is used to administer the Replication Agent. This user ID is also separate from the Replication Server maintenance user that applies replicate transactions.

A Replication Agent can use three different users:

- A user ID defined at the primary database, which the Replication Agent uses to log in to the primary data server and manipulate primary replication objects or read the database transaction log.
- A user ID that can log in to the Replication Agent and issue Replication Agent commands and configure Replication Agent parameters.
- A maintenance user ID, defined at the primary database and recorded in the primary Replication Server connection. The Replication Agent validates this user ID on behalf of the Replication Server, and the Replication Agent can be configured to ignore transactions that are created by this user ID.

DDL user processing

If DDL replication is available, this user is defined at the primary database. This user name is included in the LTL in all DDL commands sent by the Replication Agent. The DSI thread of the Replication Server uses this user name to apply the DDL to the replicate database.

Replication Agents

Sybase offers the following non-ASE Replication Agents:

- Replication Agent for DB2 UDB – provides primary data server support for a DB2 UDB server running on IBM z/OS platforms.
- Replication Agent – provides primary data server support for DB2 UDB, Microsoft SQL Server, and Oracle data servers running on Linux, UNIX, and Microsoft Windows platforms.

Replication Agent for DB2 UDB

Replication Agent for DB2 UDB product fits into a replication system as follows:

- The primary data server is DB2 UDB, which runs as a subsystem in IBM z/OS. The transaction logs are DB2 logs.
- Replication Agent for DB2 UDB runs as a started task or job in IBM z/OS. It reads the DB2 logs and retrieves the relevant DB2 active and archive log entries for the tables marked for replication for one or more DB2 subsystems. It transfers that data to Replication Server using the TCP/IP communication protocol.

The DB2 data server logs any changes to rows in DB2 tables as they occur. The information written to the transaction log includes copies of the data before and after the changes. In DB2, these records are known as “undo” and “redo” records. Control records are written for commits and aborts; These records are translated to commit and rollback operations.

The DB2 log consists of a series of data sets, which Sybase Log Extract uses to identify DB2 data changes. Because DB2 writes change records to the active log as they occur, Sybase Log Extract can process the log records immediately after they are entered.

Replication Agent

Replication Agent is a product that reads the database transaction logs in DB2 UDB, Microsoft SQL Server, or Oracle primary databases on Linux, UNIX, and Microsoft Windows platforms.

Replication Agent is implemented in the Java programming language. When you install Replication Agent, a Java Runtime Environment (JRE) is installed on the computer that is designated as the Replication Agent host machine.

Replication Agent uses the Java Database Connectivity (JDBC) protocol for all of its communication. It uses a single instance of the Sybase JDBC driver (jConnect™ for JDBC™) to manage all of its connections to Open Client and Open Server applications, including the primary Replication Server. In the case of the primary data server, Replication Agent connects to the primary database using the appropriate JDBC driver for that database.

ECDA

The Enterprise Connect Data Access (ECDA) products are Open Server-based software gateways that support DB-Library™ and CT-Library application programming interfaces (APIs), and Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC) protocols. ECDA products serve as fundamental building blocks for database middleware applications that allow you to access mainframe and LAN-based non-ASE data sources.

ECDA products provide:

- Access services that provide access to non-ASE data sources
- Administrative services (through DirectConnect Manager) that provide server-side system management

Note In Replication Server Options 15.5, you can also use ExpressConnect for Oracle to replicate to an Oracle data server.

How ECDA works

All Sybase ECDA Options provide basic connectivity to non-ASE data services. In particular, they provide access management, copy management, and remote systems management.

Each ECDA Option consists of a DirectConnect server and one or more access service libraries. The server provides the framework in which the service libraries operate. From the server, each access service library accesses data from a particular target database, such as DB2 UDB, Microsoft SQL Server, or Oracle.

Each access service library contains one or more access services that are specific sets of configuration properties. An access service transfers data between Replication Server and the target databases.

The DirectConnect server listens for, validates, and accepts incoming client connections, such as language events or remote procedure calls (RPCs). These events are routed to the target data source (replicate database) through access services, which provide target-specific connectivity features, including datatype conversion, network connectivity, and SQL transformation.

Interface file

Replication Server is an Open Server application; the preferred method for determining the location (host and port number) of another Open Server application is to look up the location in an file. The *interface* file contains a list of labels, typically server names, each of which has a corresponding host name and port number, where the identified server should be "listening" for login requests.

In the interaction between an ECDA database gateway and a Replication Server, the *interface* file is important. Because the Replication Server attempts to log in to the service identified by the server name in the Replication Server connection, that service name must exist in the Replication Server *interface* file. In addition, the *interface* file entry must also exist as a service name in the ECDA gateway configuration file entries.

A single ECDA can act as a gateway for one or many different database instances. In the ECDA configuration, each database to be accessed by the ECDA is configured as a unique *service name*. For the Replication Server to know which configured service name to connect to, it uses the server name passed at login time and expects to find a matching service name to use to complete the connection. The connection must match an *interface* file entry. For Microsoft SQL Server, the database name must be a valid database for that service. For more information about the role of service names and their configurations, refer to the *ECDA Access Service Users Guide*.

Connection shared by Replication Agent and ECDA

A single Replication Server connection can support both an ECDA gateway and a Replication Agent, because each of these components connects to the Replication Server on a different thread. If you replicate information both into and out of the same database, having a common connection for both a database gateway and a Replication Agent can make the replication system network topology less resource intensive.

To create a Replication Server connection to a database that is both primary and replicate, you must define the connection to correctly support the ECDA database gateway, then configure the Replication Agent appropriately:

- In the Replication Server, use the create connection command to define the server_name and database_name for the connection. The server_name value must match a configured service name in the ECDA.

- In the Replication Agent, set the value of the rs_source_ds parameter to that server_name, and set the value of the rs_source_db parameter to the desired database_name.

ECDA database gateways

In a Sybase replication system, the purpose of an ECDA database gateway is to apply transactions from a Replication Server to a non-ASE replicate database.

To accomplish this, Replication Server logs in to the ECDA gateway using the information specified for a Replication Server connection. Replication Server logs in to the server using the user_name and password, and issues a use database command for the database defined in the connection.

For Replication Server, there is nothing to distinguish an ECDA gateway from an Adaptive Server replicate database. Replication Server delivers the same commands—and expects the same results—from any DSI thread it communicates with.

This has the following implications:

- A valid user ID, which the Replication Server uses to log in to the replicate database, must be defined in a Replication Server connection.
- This user ID must be granted permissions to update replicate tables and execute replicate procedures.
- The replicate database must be able to maintain a RS_LASTCOMMIT table and a RS_TICKET_HISTORY table and support **rs_get_lastcommit** functionality.

  Replication Server provides sample connection profiles to set up the tables and functions required for a replicate database in DB2 UDB, Microsoft SQL Server, and Oracle databases.

  For an overview of the expectations of a replicate data server and gateway, see Chapter 6, “Replicating Data into Foreign Data Servers,” in the *Replication Server Design Guide*.
- Datatype representations must be translated to match the native datatypes of the replicate database. Replication Server provides sample connection profiles to set up the function strings, function-string classes, and base datatype definitions and translations necessary to support replication into DB2 UDB, Microsoft SQL Server, and Oracle data servers.

- The Replication Server command resume connection attempts to initiate activity with the DSI thread of the specified connection. For an ECDA, this is logging in to the DirectConnect server, accessing the RS_LASTCOMMIT table in the replicate database, and then applying transactions to the replicate database. Any failure in this sequence is recorded as a failure in the Replication Server log.

ECDA Options

There are these options available for ECDA:

- ECDA Option for ODBC
- ECDA Option for Oracle
- Mainframe Connect DirectConnect for z/OS Option

ECDA Option for ODBC

ECDA Option for ODBC provides Replication Server with an Open client interface to DB2 UDB, Microsoft SQL Server, and ODBC-accessible databases.

Note The ODBC driver for the ECDA Option for ODBC (the back-end driver connecting to the target) is not provided by Sybase; you must obtain, install, and configure it.

ECDA Option for ODBC provides access to non-ASE data sources, using the ODBC back-end (server-side) driver that you obtain for your target database, such as IBM DB2 or Microsoft SQL Server. Following the vendor's instructions, install the ODBC driver on the same server as ECDA Option for ODBC, then configure ECDA Option for ODBC to use that ODBC driver to access your database.

Note Verify that your ODBC driver is compatible with Sybase driver manager software or that it contains a driver manager.

Because ODBC drivers have varying degrees of functionality, it is important that when working with non-ASE-provided, third-party ODBC drivers, you carefully integrate and test them to be sure they meet your needs.

ECDA Option for Oracle

ECDA Option for Oracle provides Replication Server with an Open Client interface to Oracle databases. To Replication Server, ECDA Option for Oracle appears as an Open Server application that understands Oracle SQL.

Note You can also use ExpressConnect for Oracle to replicate to an Oracle data server. ExpressConnect for Oracle provides a direct interface between Replication Server and Oracle. See "ExpressConnect for Oracle" on page 49.

Mainframe Connect DirectConnect for z/OS Option

Mainframe Connect DirectConnect for z/OS Option provides Replication Server with an Open Client interface to DB2 running on a mainframe.

ExpressConnect for Oracle

ExpressConnect for Oracle (ECO) is a library that is loaded by Replication Server 15.5 or later for Oracle replication. The advantages of ECO include:

- It does not require a separate server process for starting up, monitoring, or administering.
- Since Replication Server and ECO run within the same process, no SSL is needed between them, and also there is no requirement to configure settings previously covered in the ECDA for Oracle global configuration parameters.
- Server connectivity is configured via Replication Server using the "create connection" and "alter connection" commands, thus there is no need to separately configure the equivalent to the ECDA for Oracle connect_string setting. See *Replication Server Reference Manual*.
- Configuration of settings equivalent to the ECDA for Oracle service-specific settings such as text_chunksize, autocommit, array_size is also not required, as these settings are automatically determined by Replication Server (in some cases based on the Replication Agent input) and communicated to ECO.

ECO includes certain features similar to ECDA for Oracle:

- Same set of datatype transformations.

- Language and charset conversion between Sybase data and Oracle data. In ECO, this is configured using the *map.cfg* file.
- Replication of empty strings in an ASE primary database to an Oracle replicate database, results in a string value of 1 or more (depending on whether the column is varchar or fixed char width datatype) blank spaces in Oracle.

ExpressConnect for Oracle requires only the *tnsnames.ora* file in order to establish location transparency. It does not require an *interfaces* file like ECDA for Oracle. You must specify the service name defined in the *tnsnames.ora* file for connection configuration.

See *ExpressConnect for Oracle Installation and Configuration Guide* for detailed information on ECO.

PART 2

Non-ASE Primary Data Server Topics

Chapters in this part describe data server issues and considerations for primary data servers in a replication system with non-ASE data servers.

- Chapter 3, “DB2 UDB Primary Data Server Issues for z/OS,” describes the issues specific to IBM DB2 UDB primary data server on IBM z/OS in a Sybase replication system.
- Chapter 4, “DB2 UDB Primary Data Server on UNIX, Windows, and Linux,” describes the issues specific to IBM DB2 UDB primary data server on UNIX, Windows, and Linux in a Sybase replication system.
- Chapter 5, “Microsoft SQL Server Primary Data Server Issues,” describes the issues specific to Microsoft SQL Server primary data server in a Sybase replication system.
- Chapter 6, “Oracle Primary Data Server Issues,” describes the issues specific to the Oracle primary data server in a Sybase replication system.

CHAPTER 3

DB2 UDB Primary Data Server Issues for z/OS

This chapter describes the primary data server issues and considerations specific to the DB2 UDB server on a IBM z/OS platform in a Sybase replication system.

Replication Agent for DB2 UDB

As a primary data server, the DB2 UDB interacts with the Replication Agent for DB2 UDB.

The Replication Agent identifies and transfers information about data-changing operations or transactions from a DB2 UDB primary database to a primary Replication Server.

The Replication Agent interacts with the primary Replication Server and with the RSSD of the primary Replication Server, if so configured.

Replication intrusions and impacts

The Replication Agent DB2 libraries must be authorized by the authorized program facility (APF).

The performance and operation of DB2 UDB primary data servers in a Sybase replication system might be affected as follows:

- In the DB2 UDB transaction log:
  - Replication requires a *before* and *after* image of each row that is changed. When you mark a primary table for replication, the table is altered with the DATA CAPTURE CHANGES clause. As the number of tables marked for replication increases, so does the DASD space requirement for the DB2 UDB active log data sets.
  - Using Replication Agent for DB2 UDB increases the amount of data stored in DB2 UDB logs. The size of the increase depends on the number, type, and size of the primary tables, and the types of transactions replicated. For example, update transactions require both *before* and *after* images, and they include all of the columns in a row, even if those columns do not change. For more detailed information, see the Replication Agent for DB2 UDB documentation.
- When you install the Replication Agent, two Replication Agent system tables are created in the primary DB2 UDB:
  - LTMOBJECTS contains a row for each primary table marked for replication. Its size depends on the number of tables marked for replication.
  - LTMMARKER, when updated, can be used to aid in the materialization process.
- A task started in Replication Agent for DB2 UDB can process the log of a single DB2 subsystem, or all logs in a DB2 data sharing group. This behavior is controlled by LTMCFG parameters: DataSharingOption, DataSharingMember, Log_identifier, and BSDS.
- Primary database limitations:
  - LOB replication is not supported.
  - char and varchar maximum size is 32767.
  - DDL and stored procedure replication is not supported.
- Do not use these DB2 UDB utilities, as doing so may jeopardize replication integrity:

  - LOAD LOG NO
  - RECOVER
  - REORG with RECOVER
- rs_ticket cannot be started in Replication Agent DB2 UDB. In Replication Server 15.5, it is possible to "inject" the rs_ticket into a Replication Agent DB2 connection. See *Replication Server Reference Manual* for information on sysadmin issue_ticket.

DB2 UDB primary database permissions

Create these two user IDs:

- LTMADMIN user – a TSO user, optionally named LTMADMIN, to:
  - Install, start, and stop the Replication Agent for DB2 UDB
  - Manage the Replication Agent system tables on the DB2 UDB

  The LTMADMIN user must have ALTER TABLE authority on any DB2 UDB table to be marked for replication. This user ID issues an ALTER TABLE DATA CAPTURE CHANGES command on a primary table that is marked for replication.

  The LTMADMIN user must also have TRACE, DISPLAY, and MONITOR2 permission on the DB2 UDB log files.
- Replication Server maintenance user – the user ID specified in the Replication Server create connection command for the primary database.

  Any updates applied to the primary database by the maintenance user are ignored for replication, unless the value of the LTM for z/OS LTM_process_maint_uid_trans configuration parameter is Y.

Primary data server connectivity

To connect to a primary DB2 UDB data server in an IBM z/OS environment Replication Agent for DB2 UDB requires:

- A valid user ID (the LTADMIN user identified earlier) must be defined to IBM z/OS and granted execute permission to the correct DB2 UDB plan and package. Replication Agent for DB2 UDB uses this user ID to log in to the DB2 UDB.
- Replication Agent for DB2 UDB jobs must have their Job Control Language (JCL) modified to execute with the correct accounting, user id, DB2 UDB logs, and DB2 UDB subsystem libraries.

Replication Server connectivity

Replication Agent for DB2 UDB does not use an *interface* file to connect to the Replication Server. The information needed to connect to the Replication Server is in the *LTMCFG* file. The Replication Server *interface* file does not require an entry for Replication Agent for DB2, unless the Replication Manager is used to create replication objects.

RSSD connectivity

Replication Agent for DB2 UDB does not require access to the RSSD. However, you can reduce the amount of data between the Replication Agent for DB2 UDB and Replication Server by using an RSSD.

If the LTMCFG parameter, Use_repdefs=Y, replication definitions are loaded when Replication Agent for DB2 UDB starts. If the replication definition is changed, stop and restart the Replication Agent in order for the Replication Agent to recognize the changes.

The information needed to connect to the RSSD is provided in the LTMCFG file. The parameters will all begin with RSSD, and all parameters must be entered. However, they are not verified if use_repdef is set to N.

DB2 UDB primary database configuration issues

The Replication Agent for DB2 UDB is a mainframe z/OS application consisting of two tasks that run simultaneously in a single z/OS address space:

- **Sybase Log Extract** – continuously scans the DB2 UDB active and archive logs for data-changing operations on primary tables.
- **Replication Agent for DB2 UDB for z/OS** – receives replicated transactions from Sybase Log Extract, converts them to Log Transfer Language (LTL), and sends them to the primary Replication Server.

Replication Agent can run against a single DB2 subsystem, or all logs in a DB2 data-sharing group. LTMCFG parameters describe the DB2 environment for Replication Agent for DB2 UDB (DataSharingOption, DataSharing Member, Log-identifier, and BSDS.)

When the DataSharingOption is Multi, Replication Agent for DB2 UDB refers to the Boot Strap Dataset (BSDS) parameter to identify the BSDS for each DB2 member in the data-sharing group, and displays the position of the Replication Agent for DB2 UDB and the DB2 log for each member of the group.

All Replication Agent installation and configuration issues are described in the *Replication Agent for DB2 UDB Installation Guide*. However, in a heterogeneous replication system:

- The values of the RS_source_ds and RS_source_db parameters are case-sensitive. If you do not use same case in both Replication Agent and Replication Server parameters, the connection fails.
- The Replication Agent for DB2 UDB for z/OS LTM_process_maint_uid_trans configuration parameter controls whether the Replication Agent sends transactions executed by the maintenance user to the primary Replication Server.

  In a bidirectional replication environment (replicating both into and out of the same DB2 UDB region), set the value of the LTM_process_maint_uid_trans parameter should be set to N. If you do not, transactions replicated to another site may return to be applied at the originating site, creating an endless loop.

Replication definitions for primary tables in DB2 for z/OS

The Replication Agent for DB2 UDB for z/OS Use_repdef configuration parameter controls whether the Replication Agent sends Log Transfer Language (LTL) that contains only the columns specified in a replication definition, or all of the columns in the DB2 UDB primary table.

When the value of the Use_repdef parameter is set to N, the Replication Agent sends LTL with data for all of the columns in the DB2 UDB primary table. When the value of the Use_repdef parameter is set to Y, the Replication Agent sends LTL with data for only the columns specified in the replication definition.

By sending data for only the columns needed for the replication definition, network traffic is reduced, which may improve performance.

If you set the value of Use_repdef to Y, you can use other parameters, such as suppress_col_names, to enhance Replication Agent performance. See the *Replication Agent for DB2 UDB Installation Guide*.

The LTL_table_col_case parameter controls the case in which the Replication Agent sends table and column names to Replication Server. The default in DB2 is uppercase. However, with this parameter you can change the table and column names to uppercase, lowercase, or keep the names as defined in DB2.

Names of tables can conflict with reserved words in Replication Server or the target database. To preserve the table name, you can use “with primary table named” and “with replicate table named” clauses. However, you can have Replication Agent for DB2 change the table name prior to sending the LTL to Replication Server by using the REPLICATE_NAME option in the LTMOBJECTS table. See the “DB2 table names and reserved keywords” section in Chapter 3, “Replication Agent Setup” of the Replication Agent for DB2UDB User and Troubleshooting Guide.

DB2 for z/OS primary datatype translation issues

The Replication Agent for DB2 UDB for z/OS Date_in_char, Time_in_char, and Timestamp_in_char configuration parameters control whether the Replication Agent sends values in character strings, or converts them to the Sybase datetime format.

See the *Replication Agent for DB2 UDB Users and Troubleshooting Guide* for a complete description of these parameters.

Note If you use any date- or time-related user-defined datatypes (UDDs) in a replication definition, Sybase recommends that you configure the Replication Agent to send data to the Replication Server in the format that is native to the primary database. Sybase recommends to *not* have the Replication Agent perform any datatype translations.

In general, the Replication Agent for DB2 UDB should not perform datatype translations. However, when all of the replicate data servers require the same translation, to save processing time, it is probably better to perform the translation once at the Replication Agent, rather than at each replicate database DSI.

IBM DB2 UDB represents midnight as 24.00. This format may not be compatible with other data servers. To change the value from 24.00 to 00.00, you can modify the datatype definition to automatically change the value.

IBM DB2 UDB allows year values that may be incompatible with other data servers. If the replicate data server does not allow years as early as the IBM DB2 UDB does, set the LTMCFG parameter, *Minimum_year*, so that the DB2 UDB Replication Agent modifies any year earlier than the *Minimum_year* parameter to the *Date_conv_default* parameter.

Character sets

Data within DB2 can be encoded with multiple character sets. Additionally, Replication Agent DB2 can be used to convert the replicated characters to the Replication Servers character set before it is sent to the Replication Server. The parameters that control character set properties in Replication Agent DB2 are codepage and RS_ccsid. For additional information on these parameters, see the *Replication Agent Install Guide*, Appendix titled “LTM for MVS Configuration Parameters.”

Materialization

Use Replication Agent for DB2 UDB to materialize the target with the DB2 data. The DB2 unload utility produces a data file and a punch-card file that describes the data. You can use these files as input to the materialization feature of Replication Agent for DB2 UDB to initialize the replication target. See "Using Replication Agent materialization" in Chapter 2, "Replication Server Setup," in the *Replication Server Troubleshooting Guide*.

CHAPTER 4

DB2 UDB Primary Data Server on UNIX, Windows, and Linux

This chapter describes the primary database issues and considerations specific to the DB2 UDB server on a UNIX, Windows, and Linux platform in a Sybase replication system.

Replication Agent for UDB

As a primary data server, DB2 UDB interacts with Replication Agent. An instance of the Replication Agent configured for the DB2 UDB is referred to as a *Replication Agent for UDB.*

The Replication Agent for UDB identifies and transfers information about data-changing operations or transactions from a DB2 UDB primary data server to a primary Replication Server.

Note A separate Replication Agent for UDB instance is required for each database from which transactions are replicated.

The Replication Agent interacts with the primary Replication Server and with the RSSD of the primary Replication Server, if so configured.

Note Replication Agent is a Java program. Some operating systems may require patches to support Java. Refer to the *Replication Agent Administration Guide* and the *Replication Agent Release Bulletin*.

DB2 UDB system management issues

The Replication Agent provides a number of commands that return metadata information about the primary database (database names, table names, procedure names, column names, and so on). It does this by issuing specific JDBC calls designed to return this information or by querying the system tables directly.

Replication Manager limitations

The Replication Manager plug-in cannot start, but can stop a Replication Agent instance in a primary DB2 UDB data server.

See the Replication Agent *Administration Guide* for more information about starting and stopping a Replication Agent instance.

Replication intrusions and impacts on the DB2 UDB

The performance and operation of the DB2 UDB primary data servers in a Sybase replication system might be affected by the transaction log in the following ways:

- You must set the LOGARCHMETH1 configuration parameter to LOGRETAIN or DISK:<*path*>, where <*path*> is the directory to which the logs are archived. To determine the current LOGARCHMETH1 setting, use the following UDB command:

  ```
  get db cfg for <db-alias>
  ```

- Replication requires a *before* and *after* image of each row that is changed. When you mark a primary table for replication, the Replication Agent for UDB sets the table's DATA CAPTURE option to DATA CAPTURE CHANGES. As the number of tables marked for replication increases, so does the space requirement for the DB2 UDB transaction log.
- The primary database must have a user temporary system managed tablespace with a page size of at least 8KB.

Primary database limitations in the DB2 UDB

Replication Agent does not support stored procedure or DDL replication for DB2 UDB. See the *Replication Agent Primary Database Guide*.

DB2 UDB primary database permissions

The Replication Agent for UDB requires an DB2 UDB login that has permission to access data and create new objects in the primary database.

The DB2 UDB login must have SYSADM or DBADM authority to access the primary database transaction log.

Primary data server connectivity

Replication Agent for UDB requires the following to connect to a primary DB2 UDB data server:

- If the Replication Agent for UDB is installed on a different host machine from the DB2 UDB server, install the DB2 UDB Administration Client on the Replication Agent host machine.

  If the Replication Agent for UDB software is installed on the same host machine as the DB2 UDB server, a separate DB2 UDB Administration Client is not required.

  On a Windows system, you may configure an ODBC data source in the DB2 UDB Administration Client, then use the database name and database alias specified for that ODBC data source when you configure Replication Agent for UDB connectivity.

  On a UNIX system, instead of using ODBC, catalog the node and the primary database in UDB. Then, use the database alias specified when cataloging the primary database to set the data source Replication Agent configuration parameter.

  For details on how to configure connectivity, see “Configuring IBM DB2 Universal Database connectivity” in Chapter 3, “Replication Agent for UDB,” of the *Replication Agent Primary Database Guide*.

- You can find a description of the Replication Agent configuration parameters that must be set in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Server connectivity

A description of the Replication Agent configuration parameters that must be set to allow Replication Agent to connect to the primary Replication Server in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Note Replication Agent uses TCP/IP and the Sybase JDBC driver (jConnect for JDBC, which is included in Replication Agent installation) to communicate with other Sybase servers. The Replication Agent does not rely on the Sybase *interface* file for connectivity information.

RSSD connectivity

You can find a description of the Replication Agent configuration parameters that must be set to allow Replication Agent to connect to the primary Replication Server in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Agent objects

When you initialize Replication Agent using pdb_xlog init it creates objects that support replication in the primary database. For details, see the *Replication Agent Primary Database Guide*.

Java procedure objects

Replication Agent for UDB installs *SYBRAUJAR.jar* and *SYBTRUNCJAR.jar* into the following directories:

- On Windows, the files are installed in *%DB2DIR%\SQLLIB\FUNCTION\jar\pds_username*. *%DB2DIR%* is the path to the UDB installation, and *pds_username* is the name of the primary database user specified by the pds_username Replication Agent configuration parameter.

- On UNIX, the files are installed in *$HOME/sqllib/function/jar/pds_username*. *$HOME* is the home directory of the UDB instance owner and the pds_username is the name of the primary database user specified by the pds_username Replication Agent configuration parameter.

These Jar files implement several Java procedures in the UDB primary database. Table 4-1 lists the Java procedures that are created during the Replication Agent initialization and used in log truncation.

Note If more than one Replication Agent instance is configured for a UDB server installation (one for each database from which transactions are replicated), then each Replication Agent instance must specify a different primary database user name in the pds_username configuration parameter.

Table 4-1: Java procedures for truncation

Procedure	Database name
Retrieves the name of the log file that contains the current LSN	*prefix*get_log_name_
Retrieves the version of the get_log_name Java class	*prefix*get_version_str_
Truncates the database log file or files from the archive log directory	*prefix*trunc_log_files_
Retrieves the version of the trunc_log_files Java class	*prefix*get_trunc_ver_str_

Getting actual names of the Replication objects

The Replication Agent instance generates the names of its database objects. To find the names of the Replication Agent objects, at the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

```
pdb_xlog
```

The pdb_xlog command returns a list of objects created by the Replication Agent in the primary database.

DB2 UDB primary database configuration issues

All the installation issues and configuration parameter details for a primary DB2 UDB data server are in the *Replication Agent Installation Guide*. However, this section describes additional issues specific to heterogeneous replication.

Java Runtime Environment

When you install Replication Agent, a Java Runtime Environment (JRE) that is compatible with the Replication Agent for UDB is installed. Check the *Replication Agent Release Bulletin* for any special instructions for the Java Runtime Environment.

rs_source_ds and *rs_source_db* configuration parameters

All configuration parameter values in the Replication Agent configuration file are case sensitive. Be careful when specifying the values for the rs_source_ds and rs_source_db parameters, as Replication Server is also case sensitive. If the same case is not used in both Replication Agent and Replication Server parameters, no connection occurs.

filter_maint_userid configuration parameters

The Replication Agent filter_maint_userid configuration parameter controls whether the Replication Agent forwards transactions performed by the maintenance user to the primary Replication Server. The maintenance user name is defined in the Replication Server create connection command for the primary database.

In a bidirectional replication environment (replicating both into and out of the same database), set the value of the filter_maint_userid parameter to true. If you do not, transactions replicated to another site may return to be applied at the originating site, creating an endless loop.

ltl_character_case configuration parameter

The Replication Agent ltl_character_case configuration parameter controls the character case in which the Replication Agent sends database object names to the Replication Server.

For example, if a replication definition is created for all tables named testtab, the table name sent by the Replication Agent must be testtab, or no match occurs. Because Replication Server is case sensitive, a value of TESTTAB does not match the value of testtab.

When you create replication definitions, choose a default case (for example, create all replication definitions in either all uppercase or all lowercase), and change the value of the Replication Agent ltl_character_case parameter to match.

Object names stored in uppercase

In a DB2 UDB, object names are, by default, stored in uppercase, if no case was assigned when the object was created. That means the Replication Agent sends object names in uppercase to the primary Replication Server, unless configured to do otherwise.

For more information about the ltl_character_case parameter, see the *Replication Agent Administration Guide*.

Replication definitions for primary tables in DB2 UDB

The Replication Agent use_rssd configuration parameter controls whether the Replication Agent sends Log Transfer Language (LTL) that contains only the columns specified in a replication definition, or all of the columns in the primary table.

When the value of the use_rssd parameter is false, the Replication Agent sends LTL with data for all of the columns in the primary table. When the value of the use_rssd parameter is true, the Replication Agent sends LTL with data for only the columns specified in the replication definition for each primary table.

By sending data for only the columns specified in the replication definition, network traffic is reduced, which may improve performance.

In addition, column names and parameter names are removed from the LTL because the Replication Agent can send information in the order identified by the replication definition. The LTL minimal columns and structured tokens options are also available when the value of the use_rssd parameter is true. For more information, see the *Replication Agent Administration Guide*.

DB2 UDB primary datatype translation issues

The Replication Agent allows you to control how it sends the DB2 UDB DATE, TIME, and TIMESTAMP column values to the Replication Server.

For a complete list of the DB2 UDB datatype mapping, see the *Replication Agent Primary Database Guide*.

CHAPTER 5

Microsoft SQL Server Primary Data Server Issues

This chapter describes the primary database issues and considerations specific to the Microsoft SQL Server data server in a Sybase replication system.

Note Replication Agent for Microsoft SQL Server must be installed on Microsoft Windows.

Replication Agent for Microsoft SQL Server

As a primary data server, Microsoft SQL Server interacts with Replication Agent. The Replication Agent must be installed on Microsoft Windows and must have direct access to the Microsoft SQL database log. The Replication Agent identifies and transfers information about data-changing operations or transactions from a Microsoft SQL Server primary database to a primary Replication Server.

Note A separate Replication Agent instance is required for each database from which transactions are replicated.

The Replication Agent interacts with the primary Replication Server and with the RSSD of the primary Replication Server, if so configured.

sybfilter driver

Replication Agent must be able to read Microsoft SQL Server log files. However, the Microsoft SQL Server process opens these log files with exclusive read permission, and the file cannot be read by any other processes, including Replication Agent. Before Replication Agent can replicate data, you must use the sybfilter driver to make the log files readable. See the *Replication Agent Primary Database Guide*.

Microsoft SQL Server system management issue

The Replication Agent provides a number of commands that return metadata information about the primary database (such as database names, table names, procedure names, and column names). It does this by issuing specific JDBC calls designed to return this information or by querying the system tables directly.

Replication Manager

The Replication Manager plug-in cannot start, but can stop a Replication Agent instance in a Microsoft SQL Server primary data server.

For more information about starting and stopping the Replication Agent instance, see the *Replication Agent Administration Guide*.

Replication Agent permissions

Replication Agent for Microsoft SQL Server creates database objects to assist with replication tasks in the primary database. The user ID that the Replication Agent instance uses to log in to Microsoft SQL Server must have access to the primary database. For the list of the required permissions that are automatically granted, see the *Replication Agent Primary Database Guide*.

Primary data server connectivity

Replication Agent requires a JDBC driver to communicate with the primary database. JDBC drivers for Microsoft SQL Server databases are provided by third-party database vendors. If the JDBC driver for your database is not already installed, obtain the appropriate driver from the vendor's Web site.

Refer to the *Replication Agent Release Bulletin* for the latest version of the Microsoft SQL Server JDBC.

❖ **Setting the CLASSPATH environment variable**

1 Install the JDBC driver on the host machine where Replication Agent resides or where Replication Agent can access it.

2 Add the location of the JDBC driver to the CLASSPATH environment variable:

Select Start | Settings | Control Panel | System | Environment, and add the following to the existing CLASSPATH environment variable, using the semicolon (;) as the path separator. Or create the path in the User Variables panel:

```
drive:\path_name\driver
```

where:

- *drive* is the drive letter.
- *path_name* is where you installed the JDBC driver.
- *driver* is the name of the JDBC driver. For Microsoft SQL Server, the name is *sqljdbc.jar.*

3 Click Apply, then OK.

You can find a description of the Replication Agent configuration parameters that must be set in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Server connectivity

Replication Agent uses TCP/IP and jConnect for JDBC, which is included in the Replication Agent installation to communicate with other Sybase servers. The Replication Agent does not rely on the Sybase *interface* file for connectivity information.

You can find a description of the Replication Agent configuration parameters that must be set to allow Replication Agent to connect to the primary Replication Server in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

RSSD connectivity

You can find a description of the Replication Agent configuration parameters that must be set to allow the Replication Agent to connect to the RSSD of the primary Replication Server can be found in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Agent objects in the primary database

Replication Agent creates objects in the Microsoft SQL Server primary database to assist with replication tasks. The Replication Agent objects are automatically created when you invoke the pdb_xlog command with the init keyword. The existing primary database objects can be marked for replication.

For more general information, see the *Replication Agent Administration Guide*.

Replication Agent objects

There are two variables in the transaction log component database object names:

- *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 object names. The value of the pdb_xlog_prefix_chars parameter is a list of the nonalphanumeric characters allowed in the prefix string specified by pdb_xlog_prefix. This list of allowed characters is database-specific. For example, in Microsoft SQL Server, the only nonalphanumeric characters allowed in a database object name are the $, #, @, and _ characters.

Use the pdb_xlog command to view the names of Replication Agent transaction log components in the primary database.

See the *Replication Agent Administration Guide* for details on setting up log object names.

Table objects

Insert and delete permissions are granted to Public only on the DDL shadow table for the database name *prefix*ddl_trig_xxx. No permissions are granted on other tables. Table objects that are considered Replication Agent objects are listed in the *Replication Agent Primary Database Guide*.

Procedure objects

Procedure objects that are considered Replication Agent objects are listed in the *Replication Agent Primary Database Guide*. The sp_SybSetLogforReplTable and sp_SybSetLogforReplProc procedures are created in the Microsoft SQL Server mssqlsystemresource system database. Although execute permission on these procedures is granted to Public, only the Replication Agent pds_username user can successfully execute the procedures, because only the pds_username user is granted select permission on the sys.sysschobjs table. No permissions are granted on the other procedures when they are created.

Marker objects

The marker procedures and marker shadow tables that are considered Replication Agent objects are listed in the *Replication Agent Primary Database Guide*. No permissions are granted when these procedures and tables are created.

Trigger objects

The commands that are considered Replication Agent trigger objects are listed in the *Replication Agent Primary Database Guide*.

Microsoft SQL Server primary database configuration issues

All the installation issues and configuration parameter details for a Microsoft SQL Server primary data server are in the *Replication Agent Installation Guide*. However, this section describes additional issues that are specific to heterogeneous replication.

rs_source_ds and *rs_source_db* configuration parameters

All configuration parameter values in the Replication Agent configuration file are case sensitive. Be careful when specifying the values of the rs_source_ds and rs_source_db parameters, as Replication Server is also case sensitive. If the same case is not used in both Replication Agent and Replication Server parameters, no connection occurs.

filter_maint_userid configuration parameters

If you use a Microsoft SQL Server login with sysadmin privilege as a maint_user, map the login to a user in the corresponding database, otherwise, the Replication Agent cannot correctly filter the transaction performed by this maint_user.

ltl_character_case configuration parameter

The Replication Agent ltl_character_case configuration parameter controls the character case in which the Replication Agent sends database object names to the primary Replication Server.

For example, if a replication definition is created for all tables named testtab, the table name sent by the Replication Agent must be testtab, or no match occurs. Because Replication Server is case sensitive, a value of TESTTAB does not match a value of testtab.

If you create replication definitions, choose a default case (for example, create all replication definitions in either all uppercase or all lowercase), and change the value of the Replication Agent ltl_character_case parameter to match.

The following is dependent on the collation you provided when you create the database: In a Microsoft SQL Server database, object names are stored, by default, in lowercase, if no case was assigned when the object was created. Replication Agent sends object names in lowercase to the primary Replication Server, unless configured to do otherwise.

For more information about the ltl_character_case parameter, see the *Replication Agent Administration Guide*.

Replication definitions for primary tables in Microsoft SQL Server

The Replication Agent use_rssd configuration parameter controls whether the Replication Agent sends Log Transfer Language (LTL) that contains only the columns specified in a replication definition or all of the columns in the primary table, as follows:

- When the value of the use_rssd parameter is false, the Replication Agent sends LTL with data for all of the columns in the primary table.

- When the value of the use_rssd parameter is true, the Replication Agent sends LTL with data for only the columns specified in the replication definition for each primary table.

By sending data for only the columns specified in the replication definition, network traffic is reduced, which may improve performance.

In addition, column names and parameter names are removed from the LTL because the Replication Agent can send information in the order identified by the replication definition. The LTL minimal columns and structured tokens options are also available when the value of the use_rssd parameter is true. See the *Replication Agent Administration Guide*.

To alter replication definitions, see "Replication definition change request process," in Chapter 9, "Managing Replicated Tables" in the *Replication Server Administration Guide Volume 1*.

Microsoft SQL Server primary datatype translation issues

All Microsoft SQL Server datatypes are compatible with their corresponding Adaptive Server datatypes.

varchar(max), nvarchar(max) and varbinary(max) datatypes cannot be replicated to databases other than Microsoft SQL Server.

CHAPTER 6

Oracle Primary Data Server Issues

This chapter describes the primary database issues and considerations specific to the Oracle data server in a Sybase replication system.

Replication Agent for Oracle

As a primary data server, Oracle interacts with Replication Agent. The Replication Agent identifies and transfers information about data-changing operations or transactions from an Oracle primary data server to a primary Replication Server.

Note A separate Replication Agent instance is required for each Oracle database from which transactions are replicated.

The Replication Agent interacts with the primary Replication Server and with the RSSD of the primary Replication Server, if so configured.

Note Replication Agent is a Java program. Some operating systems may require patches to support Java. Refer to the Replication Agent *Administration Guide* and the Replication Agent *Release Bulletin* for more information.

Replication definitions for primary tables in Oracle

The Replication Agent use_rssd configuration parameter controls whether the Replication Agent sends Log Transfer Language (LTL) that contains only the columns specified in a replication definition, or all of the columns in the primary table.

When the value of the use_rssd parameter is false, the Replication Agent sends LTL with data for all of the columns in the primary table. When the value of the use_rssd parameter is true, the Replication Agent sends LTL with data for only the columns specified in the replication definition for each primary table.

By sending data for only the columns specified in the replication definition, network traffic is reduced, which may improve performance.

In addition, column names and parameter names are removed from the LTL because the Replication Agent can send information in the order identified by the replication definition. The LTL minimal columns and structured tokens options are also available when the value of the use_rssd parameter is true. See the *Replication Agent Administration Guide*.

To alter replication definitions, see "Replication definition change request process," in Chapter 9, "Managing Replicated Tables" in the *Replication Server Administration Guide Volume 1*.

Replication Manager limitations

The Replication Manager plug-in cannot start, but can stop a Replication Agent instance in an Oracle primary data server.

See the *Replication Agent Administration Guide* for more information about starting and stopping the Replication Agent instance.

Oracle system management issues

The Replication Agent provides a number of commands that return metadata information about the primary database (such as database names, table names, procedure names, and column names). It does this by issuing specific JDBC calls designed to return this information, or by querying the Oracle system tables directly.

Note Oracle does not support multiple databases within a single server instance as Adaptive Server Enterprise does.

Replication intrusions and impacts in Oracle

The performance and operation of Oracle primary data servers might be affected if a Sybase replication system is incorporated. While the Replication Agent reads the Oracle online and archive redo logs to retrieve transaction information, it does require a specific log configuration. To provide and maintain the necessary information, enable these items in Oracle:

- Archiving of redo logs
- Supplemental logging of primary key and unique index data

In addition, the Replication Agent must have direct access to the Oracle redo logs and Replication Agent must run on the same platform as the primary Oracle server.

Oracle primary database permissions

The Replication Agent requires an Oracle login ID that has permission to access data and create new objects in the primary database. For a list of the Oracle login IDs that must have these required permissions, refer to the *Replication Agent Primary Database Guide.*

Note In addition to the required permissions, the operating system user who starts the Replication Agent for Oracle instance must have read access to the Oracle redo and archive logs.

Primary data server connectivity

To connect to an Oracle primary data server Replication Agent requires the following:

- The JDBC driver must be installed and referenced in the CLASSPATH system variable of the Replication Agent host machine. Java uses the contents of the CLASSPATH system variable to identify the search locations for Java classes. For the Oracle JDBC driver, the full path and file name must be included in the CLASSPATH variable, for example, *drive:\<path_name>\ojdbc14.jar*.

  For the version of the JDBC driver that is supported, see the *Replication Agent Release Bulletin*.

- For JDBC connectivity, the TNS Listener process for the Oracle primary data server must be running.
- You can find a description of the Replication Agent configuration parameters that must be set in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Server connectivity

You can find a description of the Replication Agent configuration parameters that must be set to allow Replication Agent to connect to the primary Replication Server in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Note Replication Agent uses TCP/IP and the Sybase JDBC driver (jConnect for JDBC, which is included in Replication Agent installation) to communicate with other Sybase servers. The Replication Agent does not rely on the Sybase i*nterfaces* file for connectivity information.

RSSD connectivity

You can find a description of the Replication Agent configuration parameters that must be set to allow the Replication Agent to connect to the RSSD of the primary Replication Server in Chapter 1 “Preparing for Installation” in the *Replication Agent Installation Guide*.

Replication Agent objects

There are two variables in the Replication Agent database object names:

- *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 object names, except rs_marker and rs_dump.

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, the only non-alphanumeric characters allowed in a database object name are the $, #, and _ characters.

Use the pdb_xlog command to view the names of Replication Agent transaction log components in the primary database.

See the *Replication Agent Administration Guide* for details on setting up object names.

To find the names of the objects created, at the Replication Agent administration port, invoke the pdb_xlog command with no keywords:

```
pdb_xlog
```

The pdb_xlog command returns a list of all the Replication Agent objects.

For a list of the Replication Agent for Oracle procedures and tables objects, marker objects, and sequences, refer to the *Replication Agent Primary Database Guide*.

Oracle primary database configuration issues

All the installation issues and configuration parameter details for an Oracle primary data server are provided in the *Replication Agent Installation Guide*. However, this section describes additional issues specific to heterogeneous replication.

Java Runtime Environment

When you install Replication Agent, a Java Runtime Environment (JRE) that is compatible with the Replication Agent may be installed for you. For any special instructions for the Java Runtime Environment, see the *Replication Agent Release Bulletin*.

JDBC driver required

Replication Agent requires a JDBC driver for connectivity to the primary data server. Sybase does not provide a JDBC driver for Oracle data servers. For information on how to obtain a JDBC driver for Oracle data servers, see the *Replication Agent Release Bulletin*.

rs_source_ds and *rs_source_db* configuration parameters

All configuration parameter values in the Replication Agent configuration file are case sensitive. Be careful when specifying the values of the rs_source_ds and rs_source_db parameters, as Replication Server is also case sensitive. If the same case is not used in both Replication Agent and Replication Server parameters, no connection occurs.

filter_maint_userid configuration parameters

The Replication Agent filter_maint_userid configuration parameter controls whether the Replication Agent forwards transactions performed by the maintenance user to the primary Replication Server. The maintenance user name is defined in the Replication Server create connection command for the primary database.

In a bidirectional replication environment (replicating both into and out of the same database), set the value of the filter_maint_userid parameter to the default true. If you do not, transactions replicated to another site may return to be applied at the originating site, creating an endless loop.

ltl_character_case configuration parameter

The Replication Agent ltl_character_case configuration parameter controls the case in which the Replication Agent sends database object names to the primary Replication Server.

For example, if a replication definition is created for all tables named testtab, the table name sent by the Replication Agent must be testtab, or no match occurs. Because Replication Server is case sensitive, a value of TESTTAB does not match a value of testtab.

If you create replication definitions, choose a default case (for example, create all replication definitions in either all uppercase or all lowercase), and change the value of the Replication Agent ltl_character_case parameter to match.

In an Oracle database, object names are stored, by default, in all uppercase, if no case was forced when the object was created. The Replication Agent sends object names in uppercase to the primary Replication Server, unless configured to do otherwise.

For more information about the ltl_character_case parameter, see the *Replication Agent Administration Guide*.

Oracle primary datatype translation issues

For a complete list of datatype mapping for Oracle datatypes, see "Datatype Translation and Mapping" on page 213. For more information about UDDs and their use, see the *Replication Server Administration Guide*.

Automatic Storage Management

Replication Agent for Oracle supports the use of the Oracle Automatic Storage Management (ASM) feature for online and archive redo logs. ASM provides file system and volume management support for an Oracle database environment. You can use ASM in both Real Application Cluster (RAC) and non-RAC environments. ASM provides similar benefits as a redundant array of independent disks (RAID) or a logical volume manager (LVM). Similar to those technologies, ASM allows you to define a single disk group from a collection of individual disks. ASM attempts to balance loads across all of the devices defined in the disk group. ASM also provides striping and mirroring capabilities. Unlike RAID or LVMs, ASM only supports files created and read by the Oracle database. You cannot use ASM for a general-purpose file system and cannot store binaries or flat files. The operating system cannot access ASM files.

For more information about Replication Agent support for Oracle ASM, see the *Replication Agent Primary Database Guide*.

Real Application Clusters

Replication Agent provides support for Oracle 10g and 11g Real Application Cluster (RAC) environments. When you initialize a Replication Agent for Oracle instance, the Oracle database is queried to determine how many nodes are supported by the cluster. Based on this information, Replication Agent automatically configures itself to process the redo log information from all nodes.

Note Replication of a RAC database is the same as replication from a non-RAC database.

To process the redo log data from all nodes in an Oracle RAC cluster, the Replication Agent must execute from a location that has access to the same shared storage used by the Oracle nodes to store their redo data. The Replication Agent must have read access to the shared storage where both the online and archived redo logs exist.

You can configure Replication Agent to connect to a single Oracle instance by supplying the required host, port, and Oracle SID values to the pds_host_name, pds_port_number and pds_database_name configuration parameters. In an Oracle RAC environment, Replication Agent must be able to connect to any node in the cluster in the event that a node fails or becomes unavailable. To support the configuration of multiple node locations, Replication Agent supports connectivity to all possible RAC nodes by obtaining needed information from an Oracle *tnsnames.ora* file for one specified entry. As a result, instead of configuring individual host, port, and instance names for all nodes, Replication Agent requires only the location of a tnsnames.ora file and the name of the TNS connection to use.

For more information about Replication Agent support for Oracle RAC, see the Replication Agent *Primary Database Guide.*

PART 3

Non-ASE Replicate Data Server Topics

Chapters in this part describe data server issues and considerations for replicate data servers in a replication system with non-ASE data servers.

- Chapter 7, “DB2 UDB for z/OS Replicate Data Server Issues,” describes the issues specific to DB2 UDB on IBM z/OS replicate data server in a Sybase replication system.
- Chapter 8, “DB2 UDB Replicate Data Server Issues for UNIX, Windows, and Linux,” describes the issues specific to DB2 UDB on UNIX, Windows, and Linux replicate data server in a Sybase replication system.
- Chapter 9, “Microsoft SQL Server Replicate Data Server Issues,” describes the issues specific to Microsoft SQL Server replicate data server in a Sybase replication system.
- Chapter 10, “Oracle Replicate Data Server Issues,” describes the issues specific to the Oracle replicate data server in a Sybase replication system.
- Chapter 11, “Sybase IQ Replicate Data Server Issues,” describes the issues specific to the Sybase IQ replicate data server in a Sybase replication system.

CHAPTER 7

DB2 UDB for z/OS Replicate Data Server Issues

This chapter describes only administration tasks that are unique to a Sybase replication system with non-ASE data servers. For information about basic replication system administration, see the *Replication Server Administration Guide*.

DB2 UDB for z/OS replicate data server environment

As a replicate data server in a gateway environment, DB2 UDB for z/OS interacts with the Mainframe Connect DirectConnect for z/OS Option database gateway, which accepts commands from the replicate Replication Server and applies those commands to a replicate DB2 UDB database.

DB2 UDB for z/OS system management issues

The following system management issues are specific to a replicate DB2 UDB for z/OS data server:

- The create connection command's dsi_sql_data_style parameter was used in earlier versions of Replication Server to provide some data translations for the DB2 UDB for z/OS replicate database.

  With the introduction of heterogeneous datatype support (HDS) in Replication Server version 12.0, the create connection command's dsi_sql_data_style parameter is now invalid. Do *not* use this parameter with Replication Server version 12.0 or later. The default setting should be " "(blank space).

Replication intrusions and impacts in DB2 UDB for z/OS

The only significant intrusions or impacts to the replicate DB2 UDB are the database objects created by the connection profile that creates three tables in the replicate database to support Replication Server operations:

- RS_INFO, which contains information about the sort order and character set used by the replicate database.

  Note Confirm that the INSERT statements for this table specify the proper character set and sort order for your data server.

  When using Replication Server version 12.5 or later, the replicate database sort order and character set must be recorded in the RS_INFO table. To do so, use the Replication Server rs_get_charset and rs_get_sortorder functions to retrieve the character set and sort order from the RS_INFO table in the replicate database.

- RS_LASTCOMMIT, which contains information about replicated transactions applied to the replicate database.

  Each row in the RS_LASTCOMMIT table identifies the most recent committed transaction that was distributed to the replicate database from a primary database. Replication Server uses this information to ensure that all transactions are distributed.

The Replication Server rs_get_lastcommit function retrieves information about the last transaction committed in the replicate database. For non-ASE replicate databases, rs_get_lastcommit is replaced in the database-specific function-string class by the query required to access the RS_LASTCOMMIT table in the replicate database.

- RS_TICKET_HISTORY, which contains the execution results of Replication Server command rs_ticket.

  You can issue the rs_ticket command for the primary database to measure the amount of time it takes for a command to move from the primary database to the replicate database. You can use this information to monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce. The results of each execution of rs_ticket is stored in a single row of the RS_TICKET_HISTORY table in the replicate database. You can query each row of the RS_TICKET_HISTORY table to obtain results of individual rs_ticket executions, or compare the results from different rows. The data stored in this table is not required to support replication and you may manually truncate the data in this table to reclaim space.

  Note The RS_TICKET_HISTORY table is available only in Replication Server 15.1 and later.

DB2 for z/OS replicate database permissions

To apply transactions in a replicate database, Replication Server requires a maintenance user ID that you specify in the Replication Server create connection command. The maintenance user ID must be defined to the DB2 UDB for z/OS data server and granted authority to apply transactions in the replicate database. The maintenance user ID must have permissions in the replicate DB2 UDB database:

- CREATE TABLE authority to create tables used for Replication Server processing
- UPDATE authority to all replicate tables and EXECUTE authority to all replicate stored procedures

Replicate database connectivity for DB2 UDB for z/OS

A Replication Server database connection name is made up of two parts: a data server name (server_name) and a database name (db_name).

When using the Mainframe Connect DirectConnect for z/OS Option database gateway, the server_name is the name of the database gateway server, and the db_name is the name of the replicate DB2 UDB database.

The replicate Replication Server looks for an *interface* file entry for the database gateway server_name specified in the Replication Server database connection. The replicate Replication Server logs in to the replicate data server using the user_name and password specified in the database connection.

You must make an entry in the Replication Server *interface* file to identify the host and port where the Mainframe Connect DirectConnect for z/OS Option database gateway server is listening. The *interface* file entry name must match the server_name portion of the Replication Server database connection.

Replicate database limitations in DB2 for z/OS

The following replication limitations exist with a DB2 UDB for z/OS replicate data server:

- Replication of large object (LOB) datatypes (BLOB and CLOB) is supported directly by MainframeConnect DirectConnect for z/OS Option.
- Replication Server cannot send an DB2 UDB binary value as a binary string because the MainframeConnect DirectConnect for z/OS Option database gateway performs an ASCII to EBCDIC translation on the value. Therefore, all binary or varbinary datatypes replicated to DB2 UDB for z/OS must be mapped to the rs_db2_char_for_bit or rs_db2_varchar_for_bit datatype.

DB2 for z/OS replicate database configuration issues

The heterogeneous datatype support (HDS) feature of Replication Server provides configuration information that allows you to set up the HDS feature in the replicate Replication Server and the DB2 UDB for z/OS replicate database. The configuration information is provided as part of the installation process and as part of the connection profile:

- Replication Server installation:
  - Create function strings, error classes, and user defined datatypes
- Connection profile:
  - Apply class-level datatype translations to RSSD
  - Create objects in the DB2 UDB for z/OS
  - Set connection properties
- Additional settings:
  - Settings in ECDA
  - Settings for Dynamic SQL
  - Settings for Command Batching

Replication Server installation

Replication Server installation automatically installs the required function strings and classes to support replication.

Create function strings, error classes, and user defined datatypes

The function string replaces several default Replication Server function strings with custom function strings designed to communicate with the DB2 UDB for z/OS replicate database, and access the tables and procedures that were created. These function strings are added to the Replication Server default rs_db2_function_class.

Connection profiles

Connection profiles allow you to configure your connection with a pre-defined set of properties.

Syntax

```
create connection to data_server.database
using profile connection_profile[;version]
set username [to] user
[other_create_connection_options]
[display_only]
```

Parameters

data_server – The data server that holds the database to be added to the replication system.

database – The database to be added to the replication system.

connection_profile – Indicates the connection profile that you want to use to configure a connection, modify the RSSD, and build replicate database objects.

version – Specifies the connection profile version to use.

user – The login name of the Replication Server maintenance user for the database. Replication Server uses this login name to maintain replicated data. You must specify a user name if network-based security is not enabled.

other_create_connection_options – Use the other create connection options to set connection options not specified in the profile, such as setting your password, or to override options specified in the profile, such as specifying a custom function string class to override the function string class provided in Replication Server. For a complete list of the other create connection options, see the *Replication Server Reference Manual*.

display_only – Use display_only with the using profile clause to display the commands that will be executed and the names of the servers upon which the commands will be executed. See the client and Replication Server logs for the result of using display_only.

Apply class-level datatype translations to RSSD

Class-level translations identify the primary and replicate datatypes and the replicate datatypes into which data is translated. For example, Oracle DATE should be translated to DB2 UDB replicate database TIMESTAMP.

Class-level translation is supplied for the replicate DB2 UDB for z/OS replicate database by the appropriate named connection profile:

- *rs_ase_to_db2* – translates Adaptive Server datatypes to DB2 UDB datatypes.
- *rs_udb_to_db2* – translates DB2 UDB (for UNIX and Windows) datatypes to DB2 UDB for z/OS datatypes.
- *rs_msss_to_db2* – translates Microsoft SQL Server datatypes to DB2 datatypes.

- *rs_oracle_to_db2* – translates Oracle datatypes to DB2 UDB datatypes.
- *rs_db2_connection_sample* – creates a connection to the DB2 database. (The connection may be to ECDA.)

  The connection profile provides a template for creating the Replication Server database connection for a replicate DB2 UDB for z/OS using the predefined DB2 UDB for z/OS function-string class provided with Replication Server.

Create objects in the DB2 UDB for z/OS

The connection profile creates the RS_INFO, RS_LASTCOMMIT, and the RS_TICKET_HISTORY tables in the replicate database.

Set connection properties

The connection profiles set these connection properties:

```
set error class rs_db2_error_class
set function string rs_db2_function_class
```

Additional settings

The following are additional settings to support replication.

Settings in ECDA

Use the following settings in the ECDA configuration file:

```
Transaction Mode = long
allocate = connect
SQL transformation = Sybase
```

If you are using a Mainframe Connect DirectConnect for z/OS Option database gateway for replication to a DB2 UDB for z/OS replicate database, set the following properties in the DirectConnect *db2.cfg* access service configuration file:

```
SQLTransformation=passthrough
TransactionMode=long
```

Settings for Dynamic SQL

Dynamic SQL is supported in ECDA 12.6.1 and later.

Settings for Command Batching

Command batching allows Replication Server to send multiple commands to the data server as a single command batch. You can put multiple commands in a language function-string output template, separating them with semicolons (;). If the database is configured to allow command batches, which is the default, Replication Server replaces the semicolons with that connection's DSI command separator character before sending the function string in a single batch to the data server. The separator character is defined in the dsi_cmd_separator option of the alter connection command.

If the connection to the database is not configured to allow batches, Replication Server sends the commands in the function string to the data server one at a time. To enable or disable batching for a database, use the alter connection command.

To use command batching, enter:

```
set batch = on
set dsi_cmd_separator = ;
set batch_begin = off
use_batch_markers = on
```

For information on setting batch and dsi_cmd_separator options by using the alter connection command, see the *Replication Server Reference Manual*.

CHAPTER 8

DB2 UDB Replicate Data Server Issues for UNIX, Windows, and Linux

This chapter describes administration tasks that are unique to a Sybase replication system with non-ASE data servers. For information about basic replication system administration, see the *Replication Server Administration Guide*.

DB2 UDB replicate data servers

As a replicate data server in a replication system, the DB2 UDB interacts with the ECDA Option for ODBC database gateway. ECDA Option for ODBC accepts commands from the replicate Replication Server, and applies the commands to a database residing in a DB2 UDB server.

Replication intrusions and impacts in DB2 UDB

The only significant intrusions or impacts to the DB2 UDB replicate database are the database objects that are created by the connection profile that creates two tables in the replicate database to support Replication Server operations:

- RS_INFO, which contains information about the sort order and character set used by the replicate database.

  Note Confirm that the INSERT statements for RS_INFO specify the proper character set and sort order for your DB2 UDB server.

  When using Replication Server version 12.0 or later, the replicate database sort order and character set must be recorded in the RS_INFO table.

  The Replication Server rs_get_charset and rs_get_sortorder functions retrieve the character set and sort order from the RS_INFO table in the replicate database.

- RS_LASTCOMMIT, which contains information about replicated transactions applied to the replicate database.

  Each row in the RS_LASTCOMMIT table identifies the most recent committed transaction that was distributed to the replicate database from a primary database. Replication Server uses this information to ensure that all transactions are distributed.

  The Replication Server rs_get_lastcommit function retrieves information about the last transaction committed in the replicate database. For non-ASE replicate databases, the rs_get_lastcommit function is replaced in the database-specific function-string class by the query required to access the RS_LASTCOMMIT table in the replicate database.

- RS_TICKET_HISTORY, which contains the execution results of Replication Server command rs_ticket.

You can issue the rs_ticket command for the primary database to measure the amount of time it takes for a command to move from the primary database to the replicate database. Use this information to monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce. The results of each execution of rs_ticket is stored in a single row of the RS_TICKET_HISTORY table in the replicate database. You can query each row of the RS_TICKET_HISTORY table to obtain results of individual rs_ticket executions, or compare the results from different rows. The data stored in this table is not required to support replication and you may manually truncate the data in this table to reclaim space.

Note The RS_TICKET_HISTORY table is available only in Replication Server 15.1 and later.

Limitations in the DB2 UDB replicate database

Replication of large object (LOB) datatypes (BLOB, CLOB, DBCLOB, LONG VARGRAPHIC, and LONG VARCHAR) is not supported directly from Replication Server to the ECDA Option for ODBC.

DB2 UDB replicate database permissions

To apply transactions in a replicate database, Replication Server requires a maintenance user ID that you specify using the Replication Server create connection command. The maintenance user ID must be defined at the DB2 UDB server and granted authority to apply transactions in the replicate database. The maintenance user ID must have permissions in the DB2 UDB replicate database:

- CREATE TABLE authority to create tables used for Replication Server processing
- UPDATE authority on all replicate tables

Connectivity for DB2 UDB replicate database

A Replication Server database connection name is made up of two parts: a data server name (server_name) and a database name (db_name). The server_name is the name of the ECDA Option for ODBC database gateway server, and the db_name is the name of the DB2 UDB replicate database.

The replicate Replication Server looks for an *interface* file entry for the database gateway server_name specified in the Replication Server database connection. The replicate Replication Server logs in to the replicate data server using the user_name and password specified in the database connection.

You must make an entry in the Replication Server *interface* file to identify the host and port where the ECDA Option for ODBC database gateway server is listening. The *interface* file entry name must match the server_name portion of the Replication Server database connection.

DB2 UDB replicate database configuration issues

The heterogeneous datatype support (HDS) feature of Replication Server provides configuration information that allows you to set up the HDS feature in the replicate Replication Server and the DB2 UDB replicate database. You provide this configuration information as part of the installation, and as part of the connection profile:

- Replication Server installation:
  - Create function strings, error classes, and user defined datatypes
- Connection profiles:
  - Apply class-level datatype translations to RSSD
  - Create objects in the DB2 UDB replicate database
  - Set connection properties
- Additional settings
  - Settings in ECDA (required)
  - Settings for Dynamic SQL (optional)
  - Settings for Command Batching (optional)

Replication Server installation

Replication Server installation automatically installs the required function strings and classes to support replication.

Create function strings, error classes, and user defined datatypes

The function string replaces several default Replication Server function strings with custom function strings designed to communicate with the DB2 UDB replicate database and access the tables and procedures that were created. These function strings are added to the Replication Server default rs_udb_function_class.

To find the error action defined for an error class, refer to rs_helperror in the Replication Server Reference Manual.

Connection profiles

Connection profiles allow you to configure your connection with a pre-defined set of properties.

Syntax

```
create connection to data_server.database
using profile connection_profile[;version]
set username [to] user
[other_create_connection_options]
[display_only]
```

Parameters

data_server – The data server that holds the database to be added to the replication system.

database – The database to be added to the replication system.

connection_profile – Indicates the connection profile that you want to use to configure a connection, modify the RSSD, and build replicate database objects.

version – Specifies the connection profile version to use.

user – The login name of the Replication Server maintenance user for the database. Replication Server uses this login name to maintain replicated data. You must specify a user name if network-based security is not enabled.

other_create_connection_options – Use the other create connection options to set connection options not specified in the profile, such as setting your password, or to override options specified in the profile, such as specifying a custom function string class to override the function string class provided in Replication Server. For a complete list of the other create connection options, see the *Replication Server Reference Manual*.

display_only – Use display_only with the using profile clause to display the commands that will be executed and the names of the servers upon which the commands will be executed. See the client and Replication Server logs for the result of using display_only.

Apply class-level datatype translations to RSSD

Class-level translations identify primary datatypes and the replicate datatypes into which data is to be translated (for example, Microsoft SQL Server binary should be translated to DB2 UDB CHAR FOR BIT DATA).

These connection profiles supply class-level translation for the DB2 UDB replicate database:

- *rs_ase_to_udb* – translates Adaptive Server datatypes to DB2 UDB datatypes.
- *rs_db2_to_udb* – translates DB2 for z/OS datatypes to DB2 UDB datatypes.
- *rs_msss_to_udb* – translates Microsoft SQL Server datatypes to DB2 UDB datatypes.
- *rs_oracle_to_udb* – translates Oracle datatypes to DB2 UDB datatypes.

Create objects in the DB2 UDB replicate database

The connection profile creates the RS_INFO, RS_LASTCOMMIT, and RS_TICKET_HISTORY tables in the replicate database.

Set connection properties

The connection profiles set these connection properties:

```
set error class rs_udb_error_class
set function string rs_udb_function_class
```

Additional settings

The following are additional settings to support replication.

Settings in ECDA (required)

Use the following settings in the ECDA configuration file:

```
Transaction Mode = long
allocate = connect
SQL transformation = Sybase
```

Settings for Dynamic SQL (optional)

Dynamic SQL is supported as of Replication Server 15.0.1 and requires DirectConnect UDB 12.6.1 ESD #2, or later.

Settings for Command Batching (optional)

Command batching allows Replication Server to send multiple commands to the data server as a single command batch. You can put multiple commands in a language function-string output template, separating them with semicolons (;). If the database is configured to allow command batches, which is the default, Replication Server replaces the semicolons with that connection's DSI command separator character before sending the function string in a single batch to the data server. The separator character is defined in the dsi_cmd_separator option of the alter connection command.

If the connection to the database is not configured to allow batches, Replication Server sends the commands in the function string to the data server one at a time. To enable or disable batching for a database, use the alter connection command.

To use command batching, enter:

```
set batch = on
set dsi_cmd_separator = ;
set batch_begin = off
use_batch_markers = on
```

For information on setting batch and dsi_cmd_separator options by using the alter connection command, see the *Replication Server Reference Manual*.

Using parallel DSI threads for IBM DB2 replicate database

In a heterogeneous replication environment, parallel DSI must ensure that the commit order in the replicate database is same as in the primary database. DSI can then resolve deadlock conflict, when deadlock has occurred, and Replication Server can rollback transactions and execute again.

Replication Server can maintain the order in which transactions are committed and detect conflicting updates in transactions that are simultaneously executing in parallel either:

- Internally, using Replication Server internal tables and function strings, or,
- Externally, using the rs_threads system table in the replicate database.

For external commit control, you must follow these rules:

- When different sessions operate on the same row, the update operation in session 1 should block the select operation in session 2.
- When different sessions operate on different rows, the update operation in session 1 should not block update in session 2.

Internal commit control method is better than external commit control because it depends on fewer conditions. If a deadlock occurs, the internal commit control allows Replication Server to roll back a single transaction, whereas external commit control rolls back all transactions.

Replication Server provides other options for maximizing parallelism and minimizing contention between transactions. For example, transaction serialization methods allow you to choose the degree of parallelism your system can handle without conflicts.

For detailed information on how to use parallel DSI threads, see Chapter 4, "Performance Tuning," in the *Replication Server 15.5 Administration Guide - Volume 2*.

External commit control

Replication Server can create rs_threads with row-level lock when the replicate database is IBM DB2 UDB. By default, the row level lock is "on". For example:

```
create table rs_threads (id int,seq int)
create unique index thread_index on rs_threads(id)
```

```
cluster
```

When the isolation level is 3, you must use this function string:

```
select seq from rs_threads where id = ? with cs
```

where:

cs is cursor stability, which is the default isolation level in IBM DB2 UDB.

Internal commit control

In IBM DB2 UDB, select the lock information of the current session using:

```
select agent_id from table(snapshot_lock('',-1)) as
locktable
```

To get the current session ID, use:

```
SELECT APPL.AGENT_ID FROM TABLE(SNAPSHOT_APPL_INFO('',
-1)) AS APPL WHERE APPL.APPL_ID = (VALUES
APPLICATION_ID())
```

Replication Server uses the rs_dsi_check_thread_lock function to check whether the current DSI executor thread is blocking another replicate database process. For example:

```
select count(*) as seq from table(snapshot_lock('',-1))
as T1 where TABLE_NAME!= '' AND AGENT_ID in (SELECT
AGENT_ID FROM TABLE(SNAPSHOT_APPL_INFO('',-1)) as T2
WHERE APPL_ID = (VALUES APPLICATION_ID()))
```

Transaction serialization methods

Replication Server provides four different serialization methods for specifying the level of parallelization. The serialization method you choose depends on your replication environment, and the amount of contention you expect between parallel threads. Each serialization method defines the degree to which a transaction can start before it must wait for the previous transaction to commit.

Use the dsi_partitioning_rule parameter to reduce the probability of contention without reducing the degree of parallelism assigned by the serialization method.

The serialization methods are:

- no_wait
- wait_for_start
- wait_for_commit
- wait_after_commit

Use the alter connection command with the dsi_serialization_method parameter to select the serialization method for a database connection. For example, enter the following command to select the wait_for_commit serialization method for the connection to the pubs2 database on the SYDNEY_DS data server:

```
alter connection to SYDNEY_DS.pubs2
 set dsi_serialization_method to 'wait_for_commit'
```

A transaction contains three parts:

- The beginning,
- The body of the transaction, consisting of operations such as insert, update, or delete, and
- The end of the transaction, consisting or a commit or a rollback.

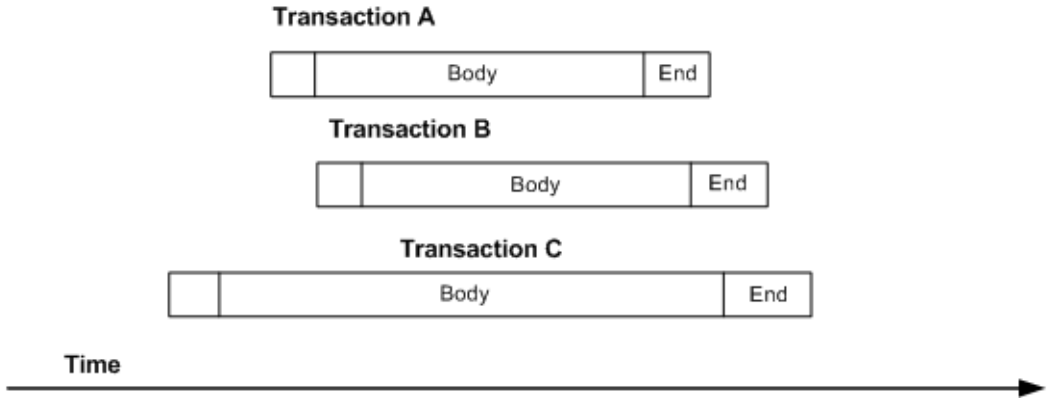
While providing commit consistency, the serialization method defines whether the beginning of the transaction waits for the previous transaction to become ready to commit or if the beginning of the transaction can be processed earlier.

no_wait

This method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit. It assumes that your primary applications are designed to avoid conflicting updates, or that dsi_partitioning_rule is used effectively to reduce or eliminate contention. Adaptive Server does not hold update locks unless dsi_isolation_level has been set to 3. The method assumes little contention between parallel transactions and results in the nearly parallel execution shown in Figure 8-3.

Note You can only set dsi_serialization_method to no_wait if dsi_commit_control is set to “on”.

Figure 8-1: Thread timing with the no_wait serialization method



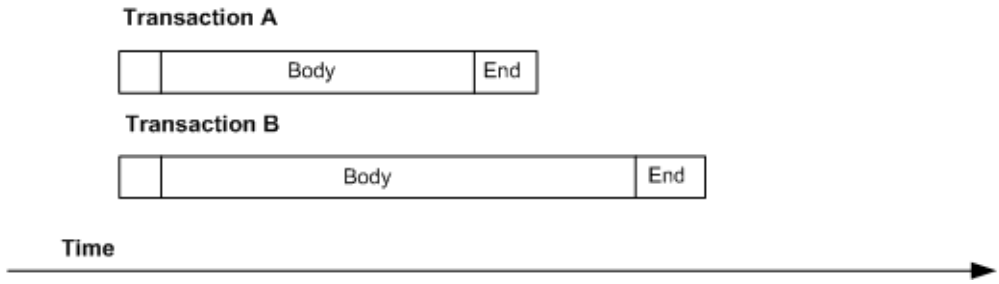
no_wait provides the better opportunity for increased performance, but also provides the greater risk of creating contentions.

wait_for_start

wait_for_start specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started. See Figure 8-2.

Sybase recommends that you do not concurrently set dsi_serialization_method to wait_for_start and dsi_commit_control to off.

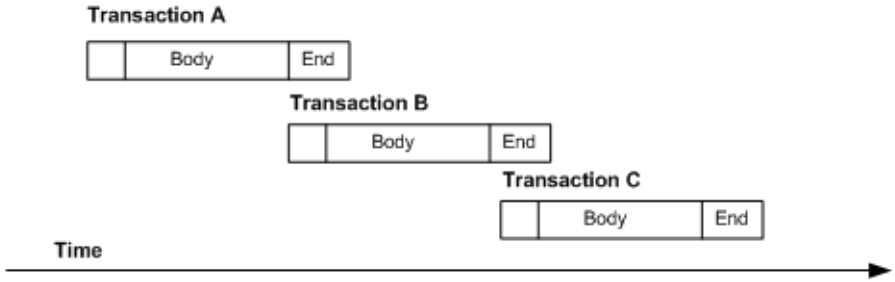
Figure 8-2: Thread timing with wait_for_start serialization method



wait_for_commit

In this method, the next thread's transaction group is not sent for processing until the previous transaction has processed successfully and the commit is being sent. This is the default setting. It assumes considerable contention between parallel transactions and results in the staggered execution shown in Figure 8-3.

Figure 8-3: Thread timing with wait_for_commit serialization method

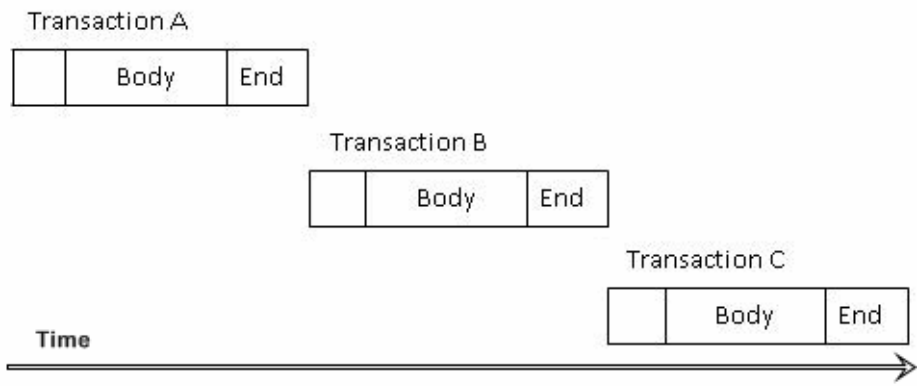


This method maintains transaction serialization by instructing the DSI to wait until a transaction is ready to commit before initiating the next transaction. The next transaction can be submitted to the replicate data server while the first transaction is committing, since the first transaction already holds the locks that it requires.

wait_after_commit

wait_after_commit specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it has committed completely. See Figure 8-4.

Figure 8-4: Thread timing with wait_after_commit serialization method



CHAPTER 9

Microsoft SQL Server Replicate Data Server Issues

This chapter describes the replicate database issues and considerations specific to the Microsoft SQL Server data server in a Sybase replication system.

Microsoft SQL Server replicate data servers

As a replicate data server, Microsoft SQL Server interacts with the ECDA Option for ODBC database gateway. The ECDA Option for ODBC server accepts commands from the replicate Replication Server, and applies those commands to a Microsoft SQL Server database.

Note The ECDA Option for ODBC supports replication of large object (LOB) datatypes (image, ntext, and text) from Replication Server directly to a Microsoft SQL Server database.

Replication intrusions and impacts on Microsoft SQL Server

The only significant intrusions or impacts to the Microsoft SQL Server replicate database are the database objects that are created by the connection profile to support Replication Server replicate database operations.

The connection profile creates three tables in the replicate database to support Replication Server operations:

- RS_INFO, which contains information about the sort order and character set used by the replicate database

  Note Confirm that the insert statements for the RS_INFO table specifies the proper character set and sort order for your Microsoft SQL Server data server.

  When using Replication Server version 12.0 or later, the replicate database sort order and character set must be recorded in the RS_INFO table.

  The Replication Server rs_get_charset and rs_get_sortorder functions retrieve the character set and sort order from the RS_INFO table in the replicate database.

- RS_LASTCOMMIT, which contains information about replicated transactions applied to the replicate database

  Each row in the RS_LASTCOMMIT table identifies the most recent committed transaction that was distributed to the replicate database from a primary database. Replication Server uses this information to ensure that all transactions are distributed.

  The Replication Server rs_get_lastcommit function retrieves information about the last transaction committed in the replicate database. For non-ASE replicate databases, the rs_get_lastcommit function is replaced in the database-specific function string class by the query required to access the RS_LASTCOMMIT table in the replicate database.

- RS_TICKET_HISTORY, which contains the execution results of Replication Server command rs_ticket.

The rs_ticket command can be issued for the primary database to measure the amount of time it takes for a command to move from the primary database to the replicate database. You can use this information to monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce. The results of each execution of rs_ticket is stored in a single row of the RS_TICKET_HISTORY table in the replicate database. You can query each row of the RS_TICKET_HISTORY table to obtain results of individual rs_ticket executions, or compare the results from different rows. The data stored in this table may be manually truncated.

Note The RS_TICKET_HISTORY table is only available in Replication Server release 15.1 and later.

Replicate database limitations on Microsoft SQL Server

The following replication limitations exist with a Microsoft SQL Server replicate data server:

- Microsoft SQL Server supports either 28 digits or 38 digits of precision, depending on the server's start-up options. The default precision is 28 digits. Replication Server does not provide user-defined datatypes (UDDs) to support the default 28 digits of precision.

  If you attempt to replicate numeric data to a Microsoft SQL Server database in excess of the server's configured precision, Replication Server returns the following error:

  ```
  E. 2007/12/14 11:14:58. ERROR #1028 DSI EXEC(134(1)
     dcm_gabeat70_devdb.devdb)
     - dsiqmint.c(2888)
     Message from server: Message: 30291, State 0,
     Severity 19 --
     '[VENDORLIB] Vendor Library Error: [[Message
     Iteration=1|Data Source Name=mssql70_devdb|
     SQLState=22003|Native Error=1007|Message=
     [Microsoft] [ODBC SQL Server Driver][SQL
     Server]The number
     '999999999999999999.999999999999999999' is out
     of the range for numeric representation (maximum
     precision 28).
  ```

```
[Message Iteration=2|SQLState=22003|Native
Error=|Message=[Microsoft][ODBC SQL Server
Driver][SQL Server]The number
'0.99999999999999999999999999999999999999' is out
of the range for numeric representation (maximum
precision 28).] <DCA>'
```

- Microsoft SQL Server supports identity columns in the same manner as Adaptive Server Enterprise, so the Replication Server function strings that set identity insert off and on work correctly with Microsoft SQL Server. However, to support 28-digit numeric precision, the Sybase native numeric datatype must be translated to the rs_msss_numeric datatype, and as a result of this translation, the identity characteristic is lost.

  If you choose to use the numeric to rs_msss_numeric datatype translation to support 28-digit precision in a Microsoft SQL Server replicate database, the replicate table cannot declare the numeric column receiving that data as an identity.

  If a replicate Microsoft SQL Server table declares a numeric column receiving translated data as an identity, Replication Server returns the following error:

  ```
  E. 2007/12/14 12:05:39. ERROR #1028 DSI EXEC(134(1)
  dcm_gabeat70_devdb.devdb)
  - dsiqmint.c(2888)
  Message from server: Message: 30291, State 0,
  Severity 19 --
  '[VENDORLIB] Vendor Library Error: [[Message
  Iteration=1|Data Source Name=mssql70_devdb|SQL
  Function=INSERT|SQLState=23000|Native
  Error=544|Message=[Microsoft][ODBC SQL Server
  Driver][SQL Server]Cannot insert explicit value
  for identity column in table 'ase_alltypes' when
  IDENTITY_INSERT is set to OFF.] <DCA>'
  ```

Microsoft SQL Server replicate database permissions

To apply transactions in a replicate database, Replication Server requires a maintenance user ID that you specify using the Replication Server create connection command. The maintenance user ID must be defined at the Microsoft SQL Server data server and granted authority to apply transactions in the replicate database. The maintenance user ID must have these permissions in the Microsoft SQL Server replicate database:

- create table authority to create tables used for Replication Server processing
- update authority on all replicate tables
- execute authority on all replicate stored procedures

Replicate database connectivity for Microsoft SQL Server

A Replication Server database connection name is made up of two parts: a data server name (server_name) and a database name (db_name). The server_name is the name of the ECDA for ODBC database gateway server, and the db_name is the name of the Microsoft SQL Server replicate database.

The replicate Replication Server looks for an *interface* file entry for the database gateway server_name specified in the Replication Server database connection. The replicate Replication Server logs in to the replicate data server using the user_name and password specified in the database connection.

Make an entry in the Replication Server *interface* file to identify the host and port where the ECDA Option for ODBC database gateway server is listening. The *interface* file entry name must match the server_name portion of the Replication Server database connection.

Microsoft SQL Server replicate database configuration

The heterogeneous datatype support (HDS) feature of Replication Server provides configuration information that allows you to set up the HDS feature in the replicate Replication Server and the Microsoft SQL Server replicate database. The configuration information is part of the installation and part of the connection profile:

- Replication Server installation:
  - Create function strings, error classes, and user defined datatypes
- Connection profile:
  - Apply class-level datatype translations to RSSD
  - Create objects in the Microsoft SQL Server database
  - Set connection properties
- Additional settings:
  - Settings in ECDA
  - Settings for Dynamic SQL
  - Settings for Command batching

Replication Server installation

Replication Server installation automatically installs the required function strings and classes to support replication.

Create function strings, error classes, and user defined datatypes

The function string replaces several default Replication Server function strings with custom function strings designed to communicate with Microsoft SQL Server and access the tables and procedures that were created. These function strings are added to the Replication Server default rs_msss_function_class.

To find the error action defined for an error class, refer to rs_helperror in the *Replication Server Reference Manual*.

Connection profiles

Connection profiles allow you to configure your connection with a pre-defined set of properties.

Syntax

```
create connection to data_server.database
using profile connection_profile[;version]
set username [to] user
[other_create_connection_options]
[display_only]
```

Parameters

data_server – The data server that holds the database to be added to the replication system.

database – The database to be added to the replication system.

connection_profile – Indicates the connection profile that you want to use to configure a connection, modify the RSSD, and build replicate database objects.

version – Specifies the connection profile version to use.

user – The login name of the Replication Server maintenance user for the database. Replication Server uses this login name to maintain replicated data. You must specify a user name if network-based security is not enabled.

other_create_connection_options – Use the other create connection options to set connection options not specified in the profile, such as setting your password, or to override options specified in the profile, such as specifying a custom function string class to override the function string class provided in Replication Server. For a complete list of the other create connection options, see the *Replication Server Reference Manual*.

display_only – Use display_only with the using profile clause to display the commands that will be executed and the names of the servers upon which the commands will be executed. See the client and Replication Server logs for the result of using display_only.

Apply class-level datatype translations to RSSD

Class-level translations identify primary datatypes and the replicate datatypes into which data is to be translated (for example, DB2 UDB TIMESTAMP should be translated to Microsoft SQL Server datetime).

Note These translations can affect Replication Server performance. Only the translations needed for your specific primary database and replicate database should be applied to the RSSD.

These connection profiles supply class-level translation for the Microsoft SQL Server replicate database:

- *rs_db2_to_msss* – translates DB2 UDB for IBM z/OS datatypes to Microsoft SQL Server datatypes.
- *rs_ase_to_msss.sql* – translates Adaptive Server datatypes to Microsoft SQL Server datatypes.
- *rs_udb_to_msss* – translates DB2 UDB (for UNIX and Windows) datatypes to Microsoft SQL Server datatypes.
- *rs_oracle_to_msss* – translates Oracle datatypes to Microsoft SQL Server datatypes.

Create objects in the Microsoft SQL Server database

The connection profile creates the RS_INFO, RS_LASTCOMMIT, and RS_TICKET_HISTORY tables in the replicate database.

Set connection properties

The connection profiles set these connection properties:

```
set error class rs_msss_error_class
set function string rs_msss_function_class
```

Additional settings

The following are additional settings to support replication.

Settings in ECDA

Use the following settings in the ECDA configuration file:

```
Transaction Mode = long
allocate = connect
SQL transformation = Sybase
```

When set batch is “on,” you must also specify:

```
DelimitSqlRequests = yes
```

If you have a tinyint datatype at the replicate table, the following parameter must be added to the Datatype Conversion section of the Microsoft SQL service in ECDA Microsoft SQL Server.

```
TinyIntResults=tinyint
```

Settings for Dynamic SQL

Dynamic SQL is supported as of Replication Server 15.0.1 and requires ECDA Option for ODBC 12.6.1 ESD #2, or later.

Settings for Command batching

Command batching allows Replication Server to send multiple commands to the data server as a single command batch. You can put multiple commands in a language function-string output template, separating them with semicolons (;). If the database is configured to allow command batches, which is the default, Replication Server replaces the semicolons with that connection's DSI command separator character before sending the function string in a single batch to the data server. The separator character is defined in the dsi_cmd_separator option of the alter connection command.

If the connection to the database is not configured to allow batches, Replication Server sends the commands in the function string to the data server one at a time. To enable or disable batching for a database, use the alter connection command.

To use command batching, enter:

```
batch = on
```

```
batch_begin = on or off
```

The use of on for batch_begin reduces the number of network transfers.

```
use_batch_markers = off
```

Additional batch markers are not required.

When set batch is "on," you must also specify the following configuration:

```
dsi_cmd_seperator set = ;
```

If you do not specify this configuration, ECDA ignores the commits after each batch , and all the replicate requests are rolled back after the dsi connection fades out.

For information on setting batch and dsi_cmd_separator options by using the alter connection command, see the *Replication Server Reference Manual*.

Using parallel DSI threads for Microsoft SQL Server replicate database

In a heterogeneous replication environment, parallel DSI must ensure that the commit order in the replicate database is same as in the primary database. DSI can then resolve deadlock conflict, when deadlock has occurred, and Replication Server can rollback transactions and execute again.

Replication Server can maintain the order in which transactions are committed and detect conflicting updates in transactions that are simultaneously executing in parallel either:

- Internally, using Replication Server internal tables and function strings,
- Externally, using the rs_threads system table in the replicate database.

For external commit control, you must follow these rules:

- When different sessions operate on the same row, the update operation in session 1 should block the select operation in session 2.
- When different sessions operate on different rows, the update operation in session 1 should not block update in session 2.

Internal commit control method is better than external commit control because it depends on fewer conditions. If a deadlock occurs, the internal commit control allows Replication Server to roll back a single transaction, whereas external commit control rolls back all transactions.

Replication Server provides other options for maximizing parallelism and minimizing contention between transactions. For example, transaction serialization methods allow you to choose the degree of parallelism your system can handle without conflicts.

For detailed information on how to use parallel DSI threads, see Chapter 4, “Performance Tuning,” in the *Replication Server 15.5 Administration Guide - Volume 2*.

External commit control

Replication Server can create rs_threads with row-level lock when the replicate database is Microsoft SQL Server. By default, the row-level lock is “on” and page level lock is “on”. For external commit control method, we need to have only row-level locking. When you apply a row-level lock to a table, you must grant unique index or primary key to that table. For example:

```
create table rs_threads
(id int,seq int CONSTRAINT PK_rs_threads PRIMARY KEY
CLUSTERED(id ASC)
WITH (ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = OFF))
```

When the isolation level is 3, use:

```
Select seq from rs_threads with(nolock) where id =?
```

For more information on selecting isolation levels for your transactions, see Chapter 4, “Performance Tuning,” in the *Replication Server 15.5 Administration Guide - Volume 2*.

Internal commit control

Replication Server uses the rs_dsi_check_thread_lock function to check whether the current DSI executor thread is blocking another replicate database process. For example:

```
select count(*) 'seq' from master..sysprocesses where
blocked = @@spid
```

Transaction serialization methods

Replication Server provides four different serialization methods for specifying the level of parallelization. The serialization method you choose depends on your replication environment, and the amount of contention you expect between parallel threads. Each serialization method defines the degree to which a transaction can start before it must wait for the previous transaction to commit.

Use the dsi_partitioning_rule parameter to reduce the probability of contention without reducing the degree of parallelism assigned by the serialization method.

The serialization methods are:

- no_wait
- wait_for_start
- wait_for_commit
- wait_after_commit

Use the alter connection command with the dsi_serialization_method parameter to select the serialization method for a database connection. For example, enter the following command to select the wait_for_commit serialization method for the connection to the pubs2 database on the SYDNEY_DS data server:

```
alter connection to SYDNEY_DS.pubs2
 set dsi_serialization_method to 'wait_for_commit'
```

A transaction contains three parts:

- The beginning,
- The body of the transaction, consisting of operations such as insert, update, or delete, and
- The end of the transaction, consisting or a commit or a rollback.

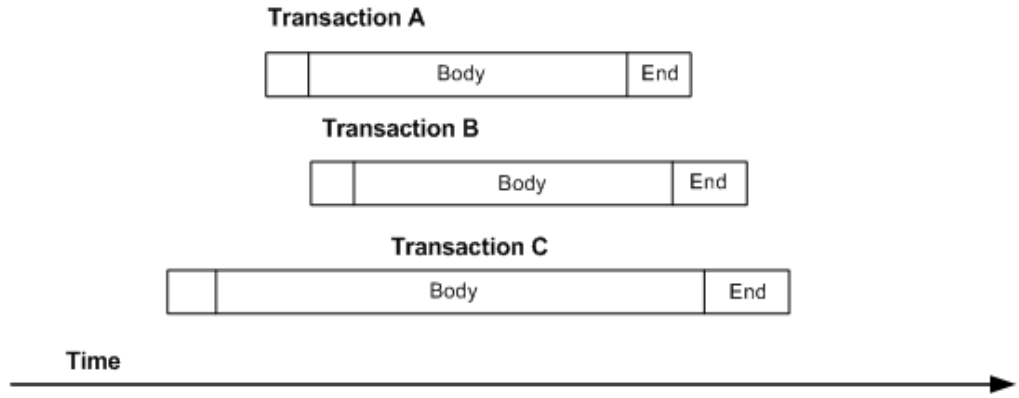
While providing commit consistency, the serialization method defines whether the beginning of the transaction waits for the previous transaction to become ready to commit or if the beginning of the transaction can be processed earlier.

no_wait

This method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit. It assumes that your primary applications are designed to avoid conflicting updates, or that dsi_partitioning_rule is used effectively to reduce or eliminate contention. Adaptive Server does not hold update locks unless dsi_isolation_level has been set to 3. The method assumes little contention between parallel transactions and results in the nearly parallel execution shown in Figure 9-3.

Note You can only set dsi_serialization_method to no_wait if dsi_commit_control is set to “on”.

Figure 9-1: Thread timing with the no_wait serialization method



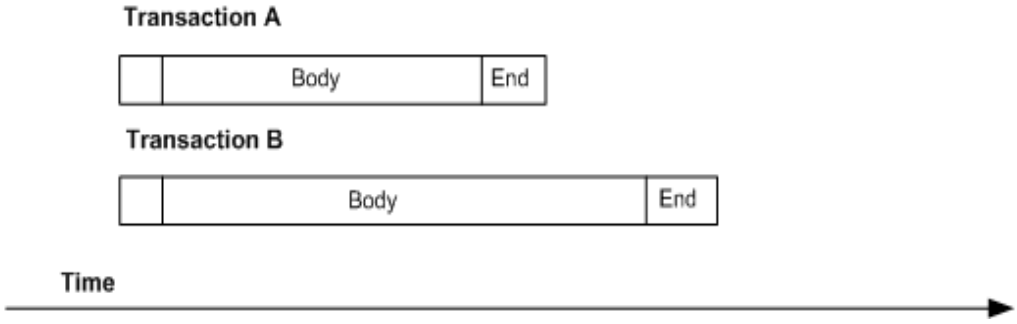
no_wait provides the better opportunity for increased performance, but also provides the greater risk of creating contentions.

wait_for_start

wait_for_start specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started. See Figure 9-2.

Sybase recommends that you do not concurrently set dsi_serialization_method to wait_for_start and dsi_commit_control to off.

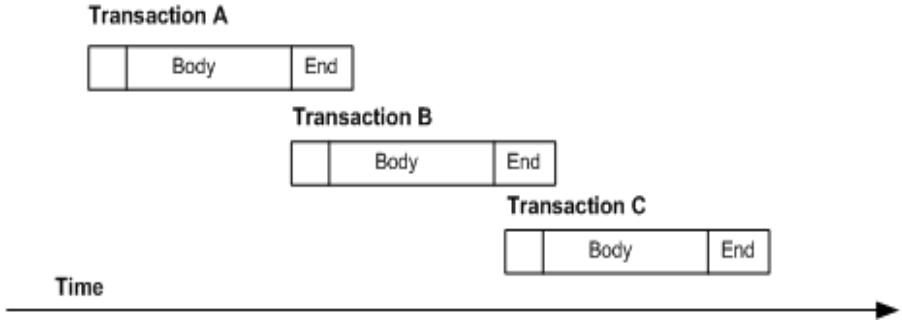
Figure 9-2: Thread timing with wait_for_start serialization method



wait_for_commit

In this method, the next thread's transaction group is not sent for processing until the previous transaction has processed successfully and the commit is being sent. This is the default setting. It assumes considerable contention between parallel transactions and results in the staggered execution shown in Figure 9-3.

Figure 9-3: Thread timing with wait_for_commit serialization method

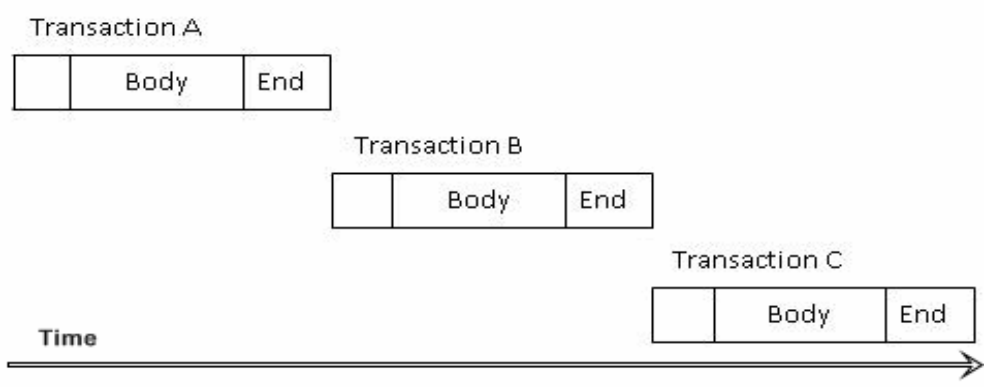


This method maintains transaction serialization by instructing the DSI to wait until a transaction is ready to commit before initiating the next transaction. The next transaction can be submitted to the replicate data server while the first transaction is committing, since the first transaction already holds the locks that it requires.

wait_after_commit

wait_after_commit specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it has committed completely. See Figure 9-4.

Figure 9-4: Thread timing with wait_after_commit serialization method



CHAPTER 10

Oracle Replicate Data Server Issues

This chapter describes the replicate database issues and considerations specific to the Oracle data server in a Sybase replication system.

Oracle replicate data servers

You can replicate to an Oracle data server, using either of:

- ECDA Option for Oracle database gateway. The ECDA Option for Oracle accepts commands from the replicate Replication Server, and applies those commands to an Oracle database.
- ExpressConnect for Oracle, which provides direct communication between Replication Server and the replicate data server. ExpressConnect for Oracle eliminates the need for installing and setting up a separate gateway server, and makes Oracle data easily accessible in a heterogeneous replication environment.

Replication intrusions and impacts on Oracle

The only significant intrusions or impacts to the Oracle replicate database are the database objects created through the connection profile that creates three tables in the replicate database to support Replication Server operations:

- RS_INFO, which contains information about the sort order and character set used by the replicate database. When using Replication Server version 12.0 or later, the replicate database sort order and character set must be recorded in the RS_INFO table.

  Note Confirm that the INSERT statements for this table specify the proper character set and sort order for your Oracle data server.

  The Replication Server rs_get_charset and rs_get_sortorder functions retrieve the character set and sort order from the RS_INFO table in the replicate database.

- RS_LASTCOMMIT, which contains information about replicated transactions applied to the replicate database. Each row in the RS_LASTCOMMIT table identifies the most recent committed transaction that was distributed to the replicate database from a primary database. Replication Server uses this information to ensure that all transactions are distributed.

  The Replication Server rs_get_lastcommit function retrieves information about the last transaction committed in the replicate database. For non-ASE replicate databases, the rs_get_lastcommit function is replaced in the database-specific function string class by the query required to access the RS_LASTCOMMIT table in the replicate database.

- RS_TICKET_HISTORY, which contains the execution results of Replication Server command rs_ticket.

  The rs_ticket command can be issued for the primary database to measure the amount of time it takes for a command to move from the primary database to the replicate database. You can use this information to monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce. The results of each execution of rs_ticket is stored in a single row of the RS_TICKET_HISTORY table in the replicate database. You can query each row of the RS_TICKET_HISTORY table to obtain results of individual rs_ticket executions, or to compare the results from different rows. The data may be manually truncated.

Note The RS_TICKET_HISTORY table is only available in Replication Server version 15.1 and later.

Oracle replicate database permissions

To apply transactions in a replicate database, Replication Server requires a maintenance user ID that you specify using the Replication Server create connection command. The maintenance user ID must be defined at the Oracle data server and granted authority to apply transactions in the replicate database. The maintenance user ID must have these permissions in the Oracle replicate database:

- CREATE TABLE authority to create tables used for Replication Server processing.
- UPDATE authority on all replicate tables and EXECUTE authority on all replicate stored procedures.

Replicate database connectivity for Oracle

Replication Server can connect to an Oracle replicate database using ECDA Option for Oracle or ExpressConnect for Oracle (ECO).

Using ECDA

A Replication Server database connection name is made up of two parts; a data server name (*server_name*) and a database name (*db_name*). The *server_name* is the name of the ECDA Option for Oracle database gateway server, and the *db_name* is the name of the Oracle SID for the replicate database.

The replicate Replication Server looks for an *interface* file entry for the database gateway *server_name* specified in the Replication Server database connection. The replicate Replication Server logs in to the replicate data server using the *user_name* and *password* specified in the database connection.

Make an entry in the Replication Server *interface* file to identify the host and port where the ECDA Option for Oracle database gateway server is listening. The *interface* file entry name must match the *server_name* portion of the Replication Server database connection.

Using ExpressConnect for Oracle

A Replication Server database connection name is made up of two parts; a data server name (*server_name*) and a database name (*db_name*). The *server_name* is the name of the desired service (Oracle instance) in the *tnsnames.ora* file. The *db_name* is the name given to the Oracle database at the time of its installation and configuration (Oracle SID). By default, this is usually "ORCL."

ExpressConnect for Oracle looks for an entry in the *tnsnames.ora* file to match the *server_name* specified in the Replication Server database connection. The replicate Replication Server logs in to the replicate data server using the *user_name* and *password* specified in the database connection.

There is no interfaces file entry required for the Oracle data server for replication using ExpressConnect for Oracle.

Oracle replicate database configuration issues

The heterogeneous datatype support (HDS) feature of Replication Server provides configuration information that allows you to set up the HDS feature in the replicate Replication Server and the Oracle replicate database. The configuration information is provided as part of the installation and as part of the connection profile:

- Replication Server installation:
  - Create function strings, error classes, and user defined datatypes
- Connection profile:
  - Apply class-level datatype translations to RSSD
  - Create objects in the Oracle replicate database
  - Set connection properties
- Additional settings:
  - ECDA settings
  - Settings for Command Batching
  - Settings for Dynamic SQL

Replication Server installation

Replication Server installation automatically installs the required function strings and classes to support replication.

Create function strings, error classes, and user defined datatypes

The function string replaces several default Replication Server function strings with custom function strings designed to communicate with an Oracle data server and access the tables and procedures. These function strings are added to the Replication Server default rs_oracle_function_class.

Connection profiles

Connection profiles allow you to configure your connection with a pre-defined set of properties. You can connect using either an ECDA Server or ExpressConnect for Oracle. Depending on whether you are connecting using an ECDA Server or ExpressConnect for Oracle, the version or option name of the connection profile should be “ecda” or “eco,” respectively.

Syntax

create connection to *data_server.database*
using profile *connection_profile*[;*version*]
set username [to] *user*
[*other_create_connection_options*]
[display_only]

Parameters

data_server – The data server that holds the database to be added to the replication system.

database – The database to be added to the replication system.

connection_profile – Indicates the connection profile that you want to use to configure a connection, modify the RSSD, and build replicate database objects.

version – Specifies the connection profile version to use.

user – The login name of the Replication Server maintenance user for the database. Replication Server uses this login name to maintain replicated data. You must specify a user name if network-based security is not enabled.

other_create_connection_options – Use the other create connection options to set connection options not specified in the profile, such as setting your password, or to override options specified in the profile, such as specifying a custom function string class to override the function string class provided in Replication Server. For a complete list of the other create connection options, see the *Replication Server Reference Manual*.

display_only – Use display_only with the using profile clause to display the commands that will be executed and the names of the servers upon which the commands will be executed. See the client and Replication Server logs for the result of using display_only.

Apply class-level datatype translations to RSSD

Class-level translations identify primary datatypes and the replicate datatypes the data should be translated into (for example, DB2 UDB TIMESTAMP should be translated to Oracle DATE).

Class-level translation is supplied for the Oracle replicate database by the appropriate named connection profile:

- *rs_ase_to_oracle* – translates Adaptive Server datatypes to Oracle datatypes.
- *rs_db2_to_oracle* – translates DB2 UDB for z/OS datatypes to Oracle datatypes.
- *rs_udb_to_oracle* – translates DB2 UDB (for UNIX and Windows) datatypes to Oracle datatypes.
- *rs_msss_to_oracle* – translates Microsoft SQL Server datatypes to Oracle datatypes.

An example of a script using ExpressConnect for Oracle version profile for an Adaptive Server Enterprise (ASE) to Oracle replication environment:

```
create connection to oracleSID_name.oracleSID_name
using profile rs_ase_to_oracle;eco
set username rs_maint_user
set password rs_maint_user_pwd
go
```

Create objects in the Oracle replicate database

The connection profile creates the RS_INFO, RS_LASTCOMMIT, and RS_TICKET_HISTORY tables in the replicate database, as well as the RS_TRIGGERS_CONTROL package.

Set connection properties

The connection profiles set these connection properties:

```
set error class rs_oracle_error_class
set function string rs_oracle_function_class
```

Additional settings

The following are additional settings to support replication.

ECDA settings

The following issues must be considered when using an Oracle replicate data server:

- In ECDA Option for Oracle version 12.0 or later, an additional trace flag allows the replicate Replication Server to control transaction commit boundaries when applying transactions to an Oracle replicate database.
- Setting the value of the ECDA autocommit trace flag to 0 (zero) in the ECDA Option for Oracle configuration file allows Replication Server to control when a COMMIT command should be sent to Oracle. When the value of the autocommit trace flag is not set, ECDA Option for Oracle commits each individual operation (INSERT, UPDATE, and DELETE) sent by the replicate Replication Server.
- Having ECDA commit each operation causes a problem at the replicate database if an error occurs in the middle of a multiple operation transaction. The replicate Replication Server may attempt to re-send the entire transaction, while ECDA has already committed each individual operation. To avoid this problem, set the value of the ECDA autocommit trace flag to 0 (zero).
- In ECDA option for Oracle 15.0 ESD#3, set the rep_sparse_parse configuration parameter to 1. This prevents SQL statements in Oracle syntax, sent from Replication Server directly to Oracle, from being parsed by ECDA Option for Oracle. This not only improves performance but is also required for using the flashback replication feature.

When rep_sparse_parse parameter is set to 0, some of the DDL and DML SQL statements sent by Replication Server are parsed and changed by ECDA option for Oracle. For example, when ECDA Option for Oracle receive a DDL statement drop table <table_name> from Replication Server, the DDL statement is parsed by ECDA Option for Oracle and changed to drop table <table_name> purge. This change should not take place if you want the replicate database recycle bin to be synchronized with the primary database recycle bin. This issue can be avoided by setting the value of rep_sparse_parse to 1.

ExpressConnect settings

Replication Server provides Oracle connection profiles, which instruct the Replication Server connection about the settings and function strings needed for appropriate database specific behaviors (such as datatype transformation, commit processing, and rs_ticket support) for an Oracle replication connection. When creating or altering a Replication Server connection to Oracle, use the appropriate Oracle connection profile (for example, the profile for ASE-to-Oracle replication or the profile for Oracle-to-Oracle replication).

Also, the replication of stored procedures in Oracle may require additional customer-provided function strings. By default, Replication Server generates ASE syntax, which may not be understood by the target database. Function strings can be added to adjust this syntax to be appropriate for the target database. For example, to transform a function call econn_test_basic_proc with one character type and one money type parameter, you must create a function string as follows:

```
create function string econn_test_basic_proc.econn_test_basic_proc for
rs_oracle_function_class with overwrite output language
'call econn_test_basic_proc(?charcolp!param?, ?moneycolp!param?)'
```

In this example, the function string causes the keyword call to be placed in front of any function replication definition and function named econn_test_basic_proc in the rs_oracle_function_class. An example of another function string that would generate a syntax acceptable to Oracle is:

```
create function string econn_test_basic_proc.econn_test_basic_proc for
rs_oracle_function_class with overwrite output language 'begin
econn_test_basic_proc(?charcolp!param?, ?moneycolp!param?);; end;;'
```

In this example, the function string prepends the same function replication definition and function with the keyword begin and appends the character string “;; end;;”

Settings for Command Batching

Command batching allows Replication Server to send multiple commands to the data server as a single command batch. You can put multiple commands in a language function-string output template, separating them with semicolons (;). If the database is configured to allow command batches, which is the default, Replication Server replaces the semicolons with that connection's DSI command separator character before sending the function string in a single batch to the data server. The separator character is defined in the dsi_cmd_separator option of the alter connection command.

If the connection to the database is not configured to allow batches, Replication Server sends the commands in the function string to the data server one at a time. To enable or disable batching for a database, use the alter connection command.

To use command batching, enter:

```
batch = on
batch_begin = off
```

When set batch is "on," you must also specify the following configuration:

```
dsi_cmd_seperator set = ;
```

As a result of a placeholder command that is used in the rs_begin function string, setting batch_begin to *on* may cause problems with starting DSI. Set batch_begin to *off* to allow the rs_begin and the rs_commit commands to be sent independently of the batches of commands, and ensures correct SQL in all transferred commands:

```
use_batch_markers = on
```

Oracle requires BEGIN and END markers for batches of commands. By configuring use_batch_markers to *on*, the markers are automatically added from the rs_batch_start and rs_batch_end function strings. For additional information on Command batching, see the *Replication Server Administration Guide*.

Settings for trigger firing

Replication Server supports disabling trigger execution for Oracle at the session or connection level. You can control trigger firing each time Replication Server executes PL/SQL commands against the replicate database. Controlling trigger execution at the replicate database eliminates data duplication and data inaccuracy errors that were caused by the absence of trigger control at the replicate database side.

To control trigger firing, RS_TRIGGER_CONTROL package has been provided, which is automatically installed when a connection to the replicate Oracle database is created through connection profiles. For every trigger to be controlled at the replicate database, re-create the trigger and add the trigger control statement at the beginning of your trigger action.

1 Set the connection parameter dsi_keep_triggers to off so that Replication Server sets the RS_TRIGGERS_CONTROL enable flag when connecting to the replicate database.

2 Add the trigger control PL/SQL code to the first line of your trigger action:

```
if RS_TRIGGER_CONTROL.IS_ENABLED then
    return;
end if;
```

This indicates that a trigger is fired by Replication Server and prevents the trigger from executing the actual application logic.

See the *Replication Server Reference Manual*.

Settings for using Oracle Flashback

Replication Agent supports Oracle Flashback at the table and transaction levels. Use Oracle flashback to query historical data, perform change analysis, and perform self-service repair to recover from logical corruptions while the database is online. Oracle customers can use flashback to undo the previous data change thereby minimizing application outages caused by operator or user errors, such as accidental deletion of valuable data, deletion of the wrong data, and dropping the wrong table. Replication Agent supports two kinds of flashback:

- Flashback a dropped table. This replicates the flashback DDL commands like drop table, flashback table to before drop, and purge recyclebin to target Oracle. To replicate purge dba_recyclebin, use DCO 15.0 ESD#3 or later, and assign the sysdba privilege to the DDL user.
- Flashback a table to a specific timestamp or SCN. This replicates the DML changes to the target Oracle database.

To flashback a table to a specific timestamp or SCN:

- Use the pdb_setreptable command to mark the table which needs to be flashbacked to a specific state.

To replicate flashback DDL statements:

- Enable recycle bin at both primary and replicate database:

  ```
  alter system set recyclebin=on
  ```

- When using ECDA, set the rep_sparse_parse parameter of the ECDA Option for Oracle to 1. The default value of this parameter is 0 when ECDA Option for Oracle 15.0 ESD #3 is used.
- Enable DDL replication by using the pdb_setrepddl enable command.

Settings for Dynamic SQL

Dynamic SQL is supported as of Replication Server 15.0.1, and requires ECDA Option for Oracle 15.0 or later, or ExpressConnect.

Using parallel DSI threads for Oracle replicate database

In a heterogeneous replication environment, parallel DSI must ensure that the commit order in the replicate database is same as in the primary database. DSI can then resolve deadlock conflict, when deadlock has occurred, and Replication Server can rollback transactions and execute again.

Replication Server can maintain the order in which transactions are committed and detect conflicting updates in transactions that are simultaneously executing in parallel either:

- Internally, using Replication Server internal tables and function strings,
- Externally, using the rs_threads system table in the replicate database.

For external commit control, you must follow these rules:

- When different sessions operate on the same row, the update operation in session 1 should block the select operation in session 2.
- When different sessions operate on different rows, the update operation in session 1 should not block update in session 2.

Internal commit control method is better than external commit control because it depends on fewer conditions. If a deadlock occurs, the internal commit control allows Replication Server to roll back a single transaction, whereas external commit control rolls back all transactions.

Replication Server provides other options for maximizing parallelism and minimizing contention between transactions. For example, transaction serialization methods allow you to choose the degree of parallelism your system can handle without conflicts.

For detailed information on how to use parallel DSI threads, see Chapter 4, "Performance Tuning," in the *Replication Server 15.5 Administration Guide - Volume 2*.

External commit control

Replication Server does not support external commit control when Oracle is the replicate database.

Internal commit control

Replication Server uses the rs_dsi_check_thread_lock function to check whether the current DSI executor thread is blocking another replicate database process. For example:

```
'select count(*) as seq from DBA_BLOCKERS
 where holding_session in (select sid from v$session
 where audsid = userenv('SESSIONID'));'
```

Transaction serialization methods

Replication Server provides four different serialization methods for specifying the level of parallelization. The serialization method you choose depends on your replication environment, and the amount of contention you expect between parallel threads. Each serialization method defines the degree to which a transaction can start before it must wait for the previous transaction to commit.

Use the dsi_partitioning_rule parameter to reduce the probability of contention without reducing the degree of parallelism assigned by the serialization method.

The serialization methods are:

- no_wait
- wait_for_start

- wait_for_commit
- wait_after_commit

Use the alter connection command with the dsi_serialization_method parameter to select the serialization method for a database connection. For example, enter the following command to select the wait_for_commit serialization method for the connection to the pubs2 database on the SYDNEY_DS data server:

```
alter connection to SYDNEY_DS.pubs2
 set dsi_serialization_method to 'wait_for_commit'
```

A transaction contains three parts:

- The beginning,
- The body of the transaction, consisting of operations such as insert, update, or delete, and
- The end of the transaction, consisting or a commit or a rollback.

While providing commit consistency, the serialization method defines whether the beginning of the transaction waits for the previous transaction to become ready to commit or if the beginning of the transaction can be processed earlier.

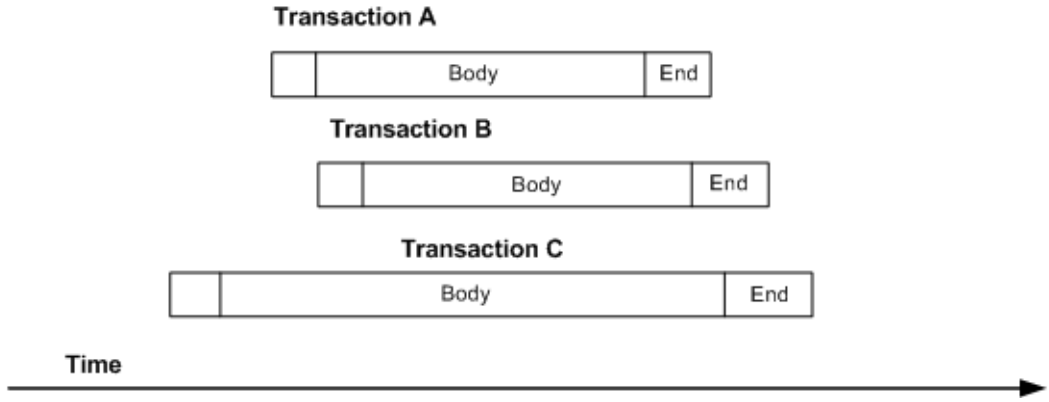
It is recommended to use wait_after_commit serialization method for those databases which use Multiversion Concurrency Control (MVCC) or Optimistic Concurrency Control such as Oracle. For all others, wait_for_commit can be used as the default method.

no_wait

This method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit. It assumes that your primary applications are designed to avoid conflicting updates, or that dsi_partitioning_rule is used effectively to reduce or eliminate contention. Adaptive Server does not hold update locks unless dsi_isolation_level has been set to 3. The method assumes little contention between parallel transactions and results in the nearly parallel execution shown in Figure 10-3.

Note You can only set dsi_serialization_method to no_wait if dsi_commit_control is set to “on”.

Figure 10-1: Thread timing with the no_wait serialization method



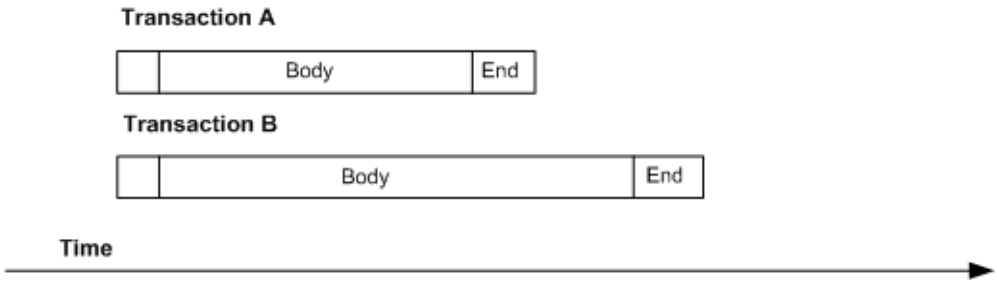
no_wait provides the better opportunity for increased performance, but also provides the greater risk of creating contentions.

wait_for_start

wait_for_start specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started. See Figure 10-2.

Sybase recommends that you do not concurrently set dsi_serialization_method to wait_for_start and dsi_commit_control to off.

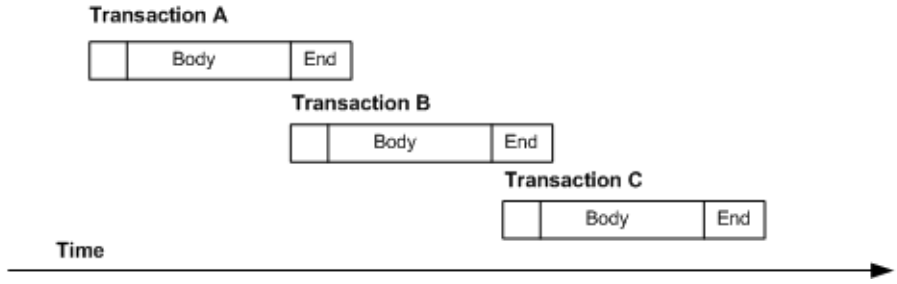
Figure 10-2: Thread timing with wait_for_start serialization method



wait_for_commit

In this method, the next thread's transaction group is not sent for processing until the previous transaction has processed successfully and the commit is being sent. This is the default setting. It assumes considerable contention between parallel transactions and results in the staggered execution shown in Figure 10-3.

Figure 10-3: Thread timing with wait_for_commit serialization method

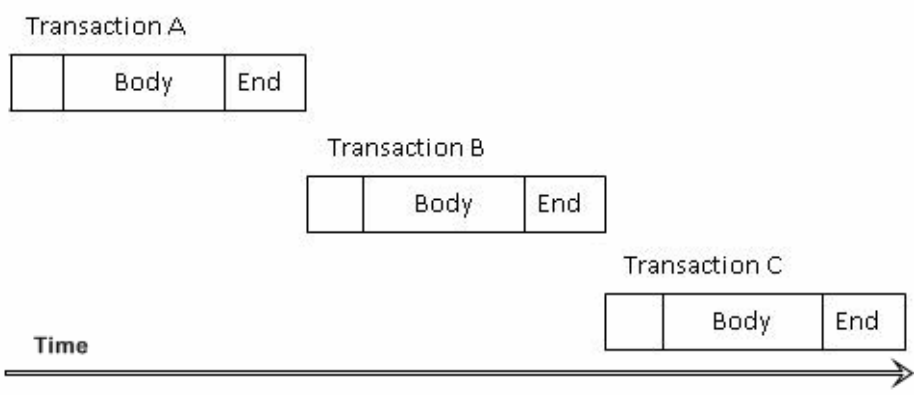


This method maintains transaction serialization by instructing the DSI to wait until a transaction is ready to commit before initiating the next transaction. The next transaction can be submitted to the replicate data server while the first transaction is committing, since the first transaction already holds the locks that it requires.

wait_after_commit

wait_after_commit specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it has committed completely. See Figure 10-4.

Figure 10-4: Thread timing with wait_after_commit serialization method



CHAPTER 11

Sybase IQ Replicate Data Server Issues

This chapter describes the replicate database issues and considerations specific to the Sybase IQ data server in a Sybase replication system and describes how to connect and configure replication to Sybase IQ from Adaptive Server.

Introduction

Sybase IQ is the ideal platform for reporting and data analysis. However, to be more effective for reporting, Sybase IQ requires real time data.

With version 15.5, Replication Server includes a real-time loading (RTL) solution for replication directly into Sybase IQ that you can use instead of the continuous replication mode that sends each logged change to the replicate database according to the primary database log-order.

Compared to the continuous replication mode, RTL achieves better performance when replicating directly into a Sybase IQ replicate database with identical database schemas by performing compilation and bulk apply.

Real-time loading solution

RTL tries to group as many compilable transactions as possible together, compiles the transactions in the group into a net change, and then uses the bulk interface in the replicate database to apply the net changes to the replicate database.

Sybase IQ provides a bulk interface that improves insert operation performance compared with the SQL language mode operation. RTL takes advantage of the Sybase IQ bulk interface to improve performance for insert as well as update and delete operations.

RTL improves performance for replication to Sybase IQ compared to the continuous replication mode and a staging solution for example, because of the:

- Reduced number of external components – leads to reduced maintenance costs and overheads since there is no requirement for the staging database.
- Reduced latency – without the overhead from the staging solution and with replication directly into Sybase IQ.
- Improved usability – the RTL configuration is straightforward and does not require function string mapping, DSI suspend and resume, data population from staging database to Sybase IQ, and the scheduling activities required for the staging solution.
- Improved replication performance – instead of sending every logged operation, RTL compilation removes the intermediate insert, update, or update operations in a group of operations and sends only the final compiled state of a replicated transaction. Depending on the transaction profile, this generally means a much smaller amount of data is processed.

See “RTL compilation and bulk apply,” for more information on RTL processing.

Licensing

Replication to Sybase IQ using real-time loading is available in the Replication Server – Real-Time Loading Edition product edition. See “Replication Server 15.5 licensing,” in Chapter 1, “New Features in Replication Server Version 15.5” in the *Replication Server 15.5 New Features Guide*.

To activate RTL, you must download the Replication Server – Real-Time Loading Edition product edition license from the Sybase Product Download Center (SPDC) at https://sybase.subscribenet.com, to which you are automatically enrolled in when you purchase a Sybase product. Log in to SPDC by using the information in your SPDC welcome e-mail. Contact your Sybase representative for more information.

Note If you have purchased your Sybase software from a Sybase reseller, you will receive a Web key rather than an e-mail message.

For information about licensing and Sybase Software Asset Management (SySAM), see the *Sybase Software Asset Management Users Guide*.

Database and platform support

RTL supports replication into Sybase IQ 12.7 ESD #3 and later.

Replication Server supports replication to Sybase IQ only from Adaptive Server version 15.0.3 or version 15.5 and later as the primary database.

64-bit support

You can achieve optimal performance using 64-bit hardware platforms. See "64-bit support," in the *Replication Server 15.5 New Features Guide*.

RTL compilation and bulk apply

During compilation, RTL rearranges data that is to be replicated by clustering the data together based on each table, and each insert, update, and delete operation, and then compiling the operations into net row operations.

RTL distinguishes different data rows by the primary key defined in a replication definition. If there is no replication definition, all columns except for text and image columns are regarded as primary keys.

For the combinations of operations found in normal replication environments, and given a table and row with identical primary keys, RTL follows these compilation rules for operations:

- An insert followed by a delete – results in no operation
- A delete followed by an insert – there is no reduction
- An update followed by a delete – results in no operation
- An insert followed by an update – results in an insert where the two operations are reduced to a final single operation. The final single operation contains the results of the first operation, overwritten by any differences in the second operation.

- An update followed by another update – results in an update where the two operations are reduced to a final single operation. The final single operation contains the results of the first operation, overwritten by any differences in the second operation.

Other combinations of operations result in invalid compilation states.

Example 1 This is an example of log-order, row-by-row changes:

```
0. create table T (k int, c int)
1. insert T values (1, 10)
2. update T set c = 11 where k = 1
3. delete T where k = 1
4. insert T values (1, 12)
5. delete T where k =1
6. insert T values (1, 13)
```

With RTL, the insert in 1 and the update in 2 can be converted to insert T values (1, 11). The converted insert and the delete in 3 cancel each other and can be removed. The insert in 4 and the delete in 5 can be removed all together. The final compiled RTL operation is the last insert in 6:

```
insert T values (1, 13)
```

Example 2 In another example of log-order, row-by-row changes:

```
1. update T set c = 14 where k = 1
2. update T set c = 15 where k = 1
3. update T set c = 16 where k = 1
```

With RTL, the update in 1 and 2 can be reduced to the update in 2. The updates in 2 and 3 can be reduced to the single update in 3 which is the net-row change of k = 1.

Replication Server uses an insert, delete, and update table in an in-memory net change database to store the net row changes which it applies to the replicate database. Net row changes are sorted by replicate table and by type of operations—insert, update, or delete—and are then ready for bulk interface.

RTL loads insert operations into the replicate table directly. Since Sybase IQ and Adaptive Server do not support bulk update and delete, RTL loads update and delete operations into temporary work tables that RTL creates inside the IQ temporary store. RTL then performs join-update or join-delete operations with the replicate tables to achieve the final result. The work tables are created and dropped dynamically.

Taking Example 2 where compilation results in `update T set c = 16 where k = 1`:

1 RTL creates the #rs_uT(k int, c int) work table.

2 RTL performs an insert into the work table with this statement:

```
insert into #rs_uT(k, c) location 'idemo.db' {select * from rs_uT}
```

3 RTL performs the join-update:

```
update T set T.c=#rs_uT.c from T,#rs_uT where T.k=#rs_uT.k
```

As RTL compiles and combines a larger number of transactions into a group, bulk operation processing improves; therefore, replication throughput and performance also improves. You can control the amount of data that RTL groups together for bulk apply by adjusting RTL sizes with the parameters in "Enabling RTL" on page 156.

There is no data loss although RTL does not apply row changes in the same order in which the changes are logged because for:

- Different data rows, the order in which the row changes are applied does not affect the result.
- The same row, applying delete before insert after compilation maintains consistency.

RTL processing and limitations

RTL applies only the net row changes of a transaction while maintaining the original commit order, and guarantees transactional consistency even as it skips intermediate row changes.This has several implications:

- If you define triggers at the replicate database, they do not fire when the RTL compiler removes rows. Triggers require every logged SQL statement to be applied. If you require table triggers to fire all the time, disable RTL compilation for that table.
- RTL does not apply row changes in the same order in which the changes are logged. To apply changes to a replicated table in log order, disable RTL compilation for that table.
- If there are referential constraints on replicate tables, you must specify the referential constraints in replication definitions.To avoid constraint errors, RTL loads tables according to replication definitions. See "Tables with referential constraints" on page 164.

- RTL for replication into Sybase IQ does not support customized function-strings or any parallel DSI serialization methods, except for the default wait_for_commit method. RTL treats customized function strings as non-compilable commands.
- RTL does not compile some types of commands called noncompilable commands and some types of tables called noncompilable tables. Replication Server reverts to log-order, row-by-row continuous replication when it encounters these commands, transactions, or tables:
  - Noncompilable commands – stored procedures, SQL statements, data definition language (DDL) transactions, system transactions, and Replication Server internal markers.
  - Noncompilable transactions – a transaction that contains noncompilable commands.
  - Noncompilable tables – tables with RTL disabled, with customized function strings, with minimal column replication enabled, and with referential constraint relationships with tables that RTL cannot compile.
- When you enable RTL by setting dsi_compile_enable to 'on' and the replication definition for the table has the replicate minimal columns clause or has the replicate_if_changed clause, Replication Server treats the table as noncompilable.

  RTL automatically changes primary-key updates to a delete followed by an insert.

  For tables without primary keys or where there are no table replication definitions, Replication Server converts updates as primary key updates to the table, as Replication Server treats all columns, except text or image columns, as primary keys.
- RTL replication attempts to group together as many transactions as possible to maximize throughput. RTL ignores parameters such as dsi_partition_rule that can stop transaction grouping.
- If errors occur during RTL processing, Replication Server retries RTL with progressively smaller transaction groups until it identifies the transaction that failed RTL compilation, and then applies the transaction using continuous replication.

To realize the performance benefits of RTL, keep the primary and replicate databases synchronized to avoid the overhead of additional processing by Replication Server when errors occur. You can set dsi_command_convert to i2di,u2di to synchronize the data although this also incurs a processing overhead. If the databases are synchronized, reset dsi_command_convert to none. See "Configuring RTL" on page 158.

- RTL performs row-count verification to ensure integrity in replication. The row count validation is based on compilation. The expected row count is the number of rows remaining after compilation.
- When there are columns with identity datatype in a replication definition, Replication Server executes this Sybase IQ command in the replicate database:
  - set temporary option identity_insert = '*table_name*' before identity column inserts and updates
  - set temporary option identity insert = "" after identity column inserts and updates

Sybase IQ replicate data servers

The replicate Replication Server interacts directly with the replicate Sybase IQ data server by logging in to the Sybase IQ replicate database and applying the replicated transactions.

Replication intrusions and impacts on Sybase IQ

The only significant intrusions or impacts to the Sybase IQ replicate database are the system tables created in the Sybase IQ replicate database through the connection profile and temporary tables created in the Sybase IQ replicate database to accommodate RTL bulk apply.

System tables

The connection profile creates three tables in Sybase IQ replicate database to support Replication Server operations:

- rs_threads, which is used by Replication Server to detect deadlocks and to perform transaction serialization between parallel DSI threads. An entry is updated in this table each time a transaction is started and more than one DSI thread is defined for a connection.
- rs_lastcommit, which contains information about replicated transactions applied to the replicate database. Each row in the rs_lastcommit table identifies the most recent committed transaction that was distributed to the replicate database from a primary database. Replication Server uses this information to ensure that all transactions are distributed.

  The Replication Server rs_get_lastcommit function retrieves information about the last transaction committed in the replicate database. For non-ASE replicate databases, the rs_get_lastcommit function is replaced in the database-specific function string class by the query required to access the rs_lastcommit table in the replicate database.
- rs_ticket_history, which contains the execution results of Replication Server command rs_ticket.

  The rs_ticket command can be issued for the primary database to measure the amount of time it takes for a command to move from the primary database to the replicate database. You can use this information to monitor Replication Server performance, module heartbeat, replication health, and table-level quiesce. The results of each execution of rs_ticket is stored in a single row of the rs_ticket_history table in the replicate database. You can query each row of the rs_ticket_history table to obtain results of individual rs_ticket executions, or to compare the results from different rows. The data may be manually truncated.

Work tables

RTL creates temporary work tables inside the IQ temporary store of the Sybase IQ database to support RTL bulk apply.The work tables are created and dropped dynamically.

The amount of space required for the temporary tables in Sybase IQ depends on the amount of the data you expect to replicate to Sybase IQ. To adjust the Sybase IQ temporary database space to accommodate the temporary work tables, use the Sybase IQ alter dbspace command. For example in:

- Sybase IQ 15.0 and later:

```
ALTER DBSPACE dbspace-name ADD FILE FileHist3
'/History1/data/file3' SIZE 500MB
```

- Sybase IQ versions earlier than 15.0 such as, Sybase IQ 12.7:

```
ALTER DBSPACE dbspace-name ADD 2 GB
```

See the Sybase IQ documentation for your version for more information.

Sybase IQ replicate database permissions

To apply transactions in a replicate database, Replication Server requires a maintenance user ID that you specify using the Replication Server create connection command.

Before replication can start, you must define the maintenance user ID at the Sybase IQ data server and grant authority to the ID to apply transactions in the replicate database. The maintenance user ID must have these permissions in the Sybase IQ replicate database:

- RESOURCE authority to create work tables and temporary indexes.
- EXECUTE permission to run the sp_iqwho stored procedure.
- GRANT ALL permission on all replicate tables.
- UPDATE authority on all replicate tables and EXECUTE authority on all replicate stored procedures.

You can grant DBA and RESOURCE authority if you are starting with a simple set up or are testing replication to Sybase IQ.

1 Create the maintenance user for Sybase IQ with relevant privileges To create the maintenance user, use the Sybase IQ *rssetup.sql* sample script:

Warning! If there is already a maintenance user ID, the script will reset the password to the default password.

```
grant connect to dbmaint identified by dbmaint
grant DBA to dbmaint
grant membership in group rs_systabgroup to dbmaint

-- Create a user for REPSRV to extract -- materialization data, etc.
```

```
-- Give sa user access to any replicated tables
-- Give sa user access to REPSRV schema
grant connect to sa identified by sysadmin
grant DBA to sa
grant membership in group rs_systabgroup to sa

-- Allow sa and dbmaint to reference replicated tables created by DBA
grant group to DBA
grant membership in group DBA to dbmaint
grant membership in group DBA to sa
go
```

You can find the sample script in the *scripts* directory within the Sybase IQ installation directory. For example, on UNIX platforms in:

- Sybase IQ versions earlier than 15.0 – */$ASDIR/scripts*
- Sybase IQ 15.0 and later – */$IQDIR15/scripts*

See the *Sybase IQ Installation and Configuration Guide* for locations of directories.

2 The IQ DBA must verify that the Sybase IQ database is compatible with Transact-SQL®. See "Transact-SQL compatibility options," in Chapter 2, "Database Options" in the *Sybase IQ Reference: Statements and Options* and Appendix A, "Compatibility with Other Sybase Databases" in the *Sybase IQ Reference: Building Blocks, Tables, and Procedures*.

3 Grant the appropriate permissions to all tables and stored procedures that are to participate in replication.

Replicate database connectivity for Sybase IQ

You do not need to use a database gateway when you use Sybase IQ as a replicate data server; the replicate Replication Server connects directly to the Sybase IQ replicate data server.

A Replication Server database connection name is made up of two parts: a data server name—server_name, and a database name—db_name. The replicate Replication Server looks for an *interface* file entry containing the Sybase IQ replicate database server_name specified in the database connection.

Use dsedit to make an entry in the Replication Server *interface* file to identify the host and port where the Sybase IQ replicate data server is listening. The *interface* file entry name must match the server_name portion of the Replication Server database connection. Restart Replication Server to activate the new entry in the Replication Server interfaces file. See "Editing the interfaces file," in Chapter 2, "Configuring Replication Server and Adding Databases with rs_init" in the *Replication Server Configuration Guide* for your platform.

Create an entry for the replicate Replication Server in the *interfaces* file of the Sybase IQ replicate server to allow Sybase IQ to connect to Replication Server and retrieve data when Replication Server sends an INSERT ... LOCATION statement to Sybase IQ.

Replication Server logs in to the Sybase IQ replicate data server using the user_name and password specified in the database connection. For Sybase IQ replicate databases, the user_name and password should be the maintenance user ID and password that you created earlier.

Sybase IQ replicate database configuration issues

This section describes configuration issues for the Sybase IQ server:

Replication Server installation

Replication Server automatically installs the required connection profile which provides function strings and classes to support replication into Sybase IQ.

Connection profiles

Connection profiles allow you to configure your connection with a pre-defined set of properties by setting the function-string class and error class, installing the user-defined datatypes (UDD) and translations for Sybase IQ, and creating the tables required for replication in the replicate Sybase IQ database.

The rs_ase_to_iq connection profile is included as part of the Replication Server installation package and it is registered when you install Replication Server. The connection profile:

- Customizes function strings, error classes, and user defined datatypes.

The function string replaces several default Replication Server function strings with custom function strings designed to communicate with a Sybase IQ data server and access the tables and procedures. These function strings are added to the Replication Server default rs_iq_function_class. RTL treats customized function strings as non-compilable commands.

- Customizes class-level datatype translations

  Class-level translations identify primary datatypes and the replicate datatypes the data should be translated into. Class-level translation is supplied for the Sybase IQ replicate database by the rs_ase_to_iq connection profile which translates Adaptive Server datatypes to Sybase IQ datatypes

- Creates tables in the Sybase IQ replicate database – rs_threads, rs_lastcommit, and rs_ticket_history tables.

- Sets connection properties – the default function-string class and error class to configure the connection to Sybase IQ:

  ```
  set error class rs_iq_error_class
  set function string rs_iq_function_class
  ```

Creating the connection to Sybase IQ

Create the connection to the replicate Sybase IQ database using create connection with the using profile clause, and specify your replicate Sybase IQ data server and database.

See "create connection with using profile clause," in Chapter 3, "Replication Server Commands" in the *Replication Server Reference Manual*.

Use admin who to verify the connection is created and Replication Server connects successfully to Sybase IQ.

Enabling RTL

After you have granted the relevant permissions and connected to the replicate Sybase IQ database as described in the earlier sections, you can enable and configure RTL for replication to Sybase IQ

Use dsi_compile_enable to enable RTL for the connection. If you set dsi_compile_enable to off, Replication Server uses continuous log-order, row-by-row replication mode. For example, set to 'off' for an affected table if replicating net row changes causes problems, such as when there is a trigger on the table which requires all the operations on that table to be replicated in log order, and therefore compilation is not allowed

Note When you set dsi_compile_enable, Replication Server disables dsi_cmd_prefetch and dsi_num_large_xact_threads.

To enable and configure RTL at the database level to affect only the specified database:

```
alter connection to IQ_data_server.iq_database
set dsi_compile_enable to 'on'
go
```

You can also enable and configure RTL at the server or table levels:

- Server level – affects all database connections to Replication Server:

  ```
  configure replication server
  set dsi_compile_enable to 'on'
  ```

- Table level – affects only the replicate tables you specify. If you specify a parameter at both the table-level and database level, the table level parameter takes precedence over the database-level parameter. If you do not specify a table level parameter, the setting for the parameter applies at the database level. To set a parameter for a table, use alter connection and the for replicate table named clause, for example:

  ```
  alter connection to IQ_data_server.iq_database
  for replicate table named dbo.table_name
  set dsi_compile_enable to 'on'
  ```

  Using the for replicate table name clause alters connection configuration at the table level. The configuration changes apply to replicate data from all the subscriptions and all the replication definitions of the tables you specify.

  Note For table-level configuration you can use only alter connection, as Replication Server does not support the for clause with create connection.

After you execute dsi_compile_enable, suspend and resume the connection to the replicate Sybase IQ database:

Configuring RTL

Once you alter the connection to Sybase IQ and set dsi_compile_enable to on, Replication Server uses these parameters to determine when to start and stop transaction grouping and compilation:

- dsi_compile_max_cmds – specifies, in number of commands, the maximum size of a group of transactions. When RTL reaches the maximum group size for the current group that it is compiling, RTL starts a new group.

  If there is no more data to read, and even if the group does not reach the maximum number of commands, RTL completes grouping the current set of transactions into the current group. The default is 100,000 commands.

  dsi_bulk_threshold – specifies the number of net row change commands after compilation has occurred on a table for a command type, that when reached, triggers Replication Server to use bulk copy-in on that table for the same command type. The default is 20 net row change commands.

  dsi_command_convert – specifies how to convert a replicate command. A combination of these operations specifies the type of conversion:

  - d – delete
  - i – insert
  - u – update
  - t - truncate
  - none – no operation

  Combinations of operations for dsi_command_convert include i2none, u2none, d2none, i2di, t2none, and u2di. The operation before conversion precedes the “2” and the operations after conversion are after the “2”. For example:

  - d2none – do not replicate the delete command. With this option, you do not need to customize the rs_delete function string if you do not want to replicate delete operations.
  - i2di,u2di – convert both insert and update to delete followed by insert, which is equivalent to an autocorrection.
  - t2none – do not replicate truncate table command.

  The default for dsi_command_convert is none which means there is no command conversion.

RTL automatically sets the Sybase-recommended values for dsi_compile_max_cmds, dsi_bulk_threshold, and dsi_command_convert, which are the default values. However, you can specify your own values to tune performance in your replication environment. For example, if you want to adjust these values from the default values when you enable RTL:

```
alter connection to IQSRVR.iqdb
set dsi_compile_enable to 'on'
set dsi_compile_max_commands to '50000'
set dsi_bulk_threshold to '15'
set dsi_command_convert to 'i2di,u2di'
go
```

See "alter connection," in Chapter 3, "Replication Server Commands" in the *Replication Server Reference Manual* for full descriptions of these parameters.

System table support in Replication Server

Replication Server uses the rs_tbconfig table to store table-level configuration parameters, and the ref_objowner and ref_objname columns in the rs_columns table to support referential constraints.

See Chapter 8, "Replication Server System Tables" in the *Replication Server Reference Manual* for full table descriptions.

Scenario for replication to Sybase IQ

The following sections describe a scenario to set up replication to Sybase IQ using RTL and to test that replication works.

The Adaptive Server database administrator (ASE DBA), the Sybase IQ database administrator (IQ DBA), and you, as the replication system administrator (RSA), must prepare Adaptive Server, Replication Server, and Sybase IQ for replication and set up the connection to the Sybase IQ database:

In this scenario, dbo is the table owner of the testtab table in the pdb1 database of the ASE_DS primary Adaptive Server. c1, c2, and c3 are columns in testtab with int, int, and char(10) datatypes respectively, and IQSRVR is the replicate Sybase IQ data server containing the iqdb database.

Creating test tables

To test that replication works in these examples, create the testtab table in the primary and replicate databases and grant permissions to testtab logged in as the maintenance user.

1 At the primary database pdb1 in Adaptive Server, create the table

```
use pdb1
go
create table dbo.testtab(c1 int primary key, c2 int,
c3 char(10))
go
```

2 At the replicate database iqdb in the IQSRVR Sybase IQ data server:

```
use iqdb
go
create table dbo.testtab(c1 int primary key, c2 int,
c3 char(10))
go
grant all on dbo.testtab to public
go
```

Create the connection to the primary and replicate databases

1 Use the Replication Server rs_init utility to create the Replication Server connection to the Adaptive Server database that will act as the primary database. See the *Replication Server Configuration Guide* to use rs_init.

2 Create the connection to the replicate Sybase IQ database.

Note You cannot use rs_init to create the connection to Sybase IQ.

This example uses the iqdb database in the IQSRVR data server, and the default dbmaint Sybase IQ maintenance user.

```
create connection to IQSRVR.iqdb
using profile rs_ase_to_iq;standard
set username to dbmaint
set password to dbmaint
go
```

If the command is successful, you see:

```
Connection to 'IQSRVR.iqdb' is created.
```

See "create connection with using profile clause," in Chapter 3, "Replication Server Commands" in the *Replication Server Reference Manual*.

Verify that the connection is running:

```
admin who
go
```

If the connection is running, you see:

```
Spid   Name       State              Info
----   --------   ----------------   -----------------
 63    DSI EXEC   Awaiting Command   103(1)IQSRVR.iqdb
 62    DSI        Awaiting Message   103 IQSRVR.iqdb
 35    SQM        Awaiting Message   103:0 IQSRVR.iqdb
```

Enable RTL

1 To enable and configure RTL at the database-level to affect only the specified database:

```
alter connection to IQSRVR.iqdb
set dsi_compile_enable to 'on'
go
```

2 Suspend and resume the connection to the replicate Sybase IQ database to enable the change to the connection. To suspend the connection:

```
suspend connection to IQSRVR.iqdb
go
```

To resume the connection:

```
resume connection to IQSRVR.iqdb
go
```

Marking tables to prepare for replication testing

This section describes how to mark tables in the primary Adaptive Server database that you want to replicate to the Sybase IQ database.

In these examples, dbo is the table owner of testtab in the pdb1 database of the ASE_DS primary Adaptive Server. c1, c2, and c3 are columns in testtab with int, int, and char(10) datatypes, respectively.

1 Before you mark the tables, insert data rows into testtab for testing replication:

```
insert into testtab values(1,1,'testrow 1')
insert into testtab values(2,2,'testrow 2')
insert into testtab values(3,3,'testrow 3')
go
```

If the inserts are successful, you see:

```
(1 row affected)
(1 row affected)
(1 row affected)
```

2 Mark testtab for replication:

- ASE 15.0.3 and later:

```
sp_setrepdefmode testtab,'owner_on'
go
```

- Versions earlier than ASE 15.0.3:

```
sp_setreptable testtab,'true', 'owner_on'
go
```

You see:

```
The replication status for 'testtab' is set to true,
owner_on.
```

Creating replication definitions and subscriptions

After you enable and configure RTL, create replication definitions and subscriptions for the tables that you marked for replication to Sybase IQ.

1 Create the repdef_testtab replication definition:

```
create replication definition repdef_testtab
with primary at ASE_DS.pdb1
with primary table named 'testtab'
with replicate table named dbo.'testtab'
(c1 int, c2 int, c3 char(10))
primary key(c1)
```

```
go
```

Add any required referential constraint clauses to the replication definitions to support RTL. See"Tables with referential constraints" on page 164.

2 Create subscriptions to match each of the table and stored procedure replication definitions:

```
create subscription sub_testtab for repdef_testtab
with replicate at IQSRVR.iqdb
go
```

3 Verify that testtab is materialized by logging in to Sybase IQ and executing:

```
select * from dbo.testtab
go
```

If materialization is successful, you see:

```
c1          c2          c3
---------   ---------   ---------
1           1           testrow 1
2           2           testrow 2
3           3           testrow 3
(3 rows affected)
```

Verifying that RTL works

1 Log in to the primary Adaptive Server and execute some DML operations, such as inserting new rows into testtab:

```
insert into testtab values(4,4,'testrow 4')
insert into testtab values(5,5,'testrow 5')
insert into testtab values(6,6,'testrow 6')
go
```

This should be the output:

```
(1 row affected)
(1 row affected)
(1 row affected)
```

2 Log in to Sybase IQ and verify that the changes to testtab have replicated to the Sybase IQ database:

```
select * from dbo.testtab
go
```

If replication is successful, this is the output:

```
c1          c2          c3
---------   ---------   ---------
1           1           testrow 1
2           2           testrow 2
3           3           testrow 3
4           4           testrow 4
5           5           testrow 5
6           6           testrow 6
(6 rows affected)
```

Tables with referential constraints

You can use a replication definition to specify tables that have referential constraints, such as a foreign key and other check constraints, so that RTL is aware of these tables. See “RTL processing and limitations” on page 149.

Usually, the referencing table contains referential constraints for a referenced table within the same primary database. However, RTL extends referential constraints support to referenced tables from multiple primary databases. You can specify the referencing table in replication definition for each primary database. However, if multiple referential constraints conflict with each other, Replication Server randomly selects one referential constraint.

Creating and altering replication definitions

Use the create replication definition command with the references parameter to specify the table with referential constraints.

create replication definition
...
(*column_name* [as *replicate_column_name*]
...
[map to *published_datatype*]] [quoted]

```
    [references [table_owner.]table_name [(column_name)]] …)
  ....]
```

Use the alter replication definition command with the references parameter to add or change a referencing table. Use the null option to drop a reference.

```
alter replication definition
  .....
add column_name [as replicate_column_name]
[map to published_datatype] [quoted]
[references [table_owner.]table_name [(column_name)]
...
| alter columns with column_name references
{[table_owner.]table_name [(column_name)] | NULL}
[, column_name references {[table_owner.]table_name [(column_name)]
| NULL}
    ...
```

For both alter replication definition and create replication definition with the reference clause, Replication Server:

- Treats the reference clause as a column property. Each column can reference only one table.
- Does not process the column name you provide in the *column_name* parameter within the reference clause.
- Does not allow referential constraints with cyclical references. For example, the original referenced table cannot have a referential constraint to the original referencing table.

During replication processing, RTL loads:

- Inserts to the referenced tables before the referencing table you specify in the replication definition.
- Deletes to the referenced tables after the table you specify in the replication definition.

In some cases, updates to both tables fail because of conflicts. To prevent RTL from retrying replication processing, and thus decreasing performance, you can:

- Stop replication updates by setting dsi_command_convert to "u2di," which converts updates to deletes and inserts.
- Turn off dsi_compile_enable to avoid compiling the affected tables.

RTL cannot compile tables with customized function strings, and tables that have referential constraints to an existing table that it cannot compile. By marking out these tables, RTL optimizes replication processing by avoiding transaction retries due to referential constraint errors.

Displaying RTL information

You can display information on table-level configuration parameters and table references.

Displaying table-level configuration parameters

Use the Replication Server admin config command to display information about table-level configuration parameters.

For example, to display all configuration parameters after using dsi_command_convert to set d2none on the tb1 table in the iqdb database of the IQSRVR data server:

```
admin config, "table", IQSRVR, iqdb
```

See "admin config," in Chapter 3, "Replication Server Commands" in the *Replication Server Reference Manual* for syntax, parameters, and more examples.

Displaying table references

Use the rs_helprep stored procedure which you can execute on the Replication Server System Database (RSSD) to display information about table references and RTL information.

For example, to display information about the "authors" replication definition created using create replication definition, enter:

```
rs_helprep authors
```

See "rs_helprep," in Chapter 6, "RSSD Stored Procedures" in the *Replication Server Reference Manual* for syntax, parameters, and more examples.

Displaying the net-change database

Replication Server has a net-change database that acts as an in-memory repository for storing the net row changes of a transaction, that is, the compiled transaction. There is one net-change database instance for each transaction. Replication Server allows external processes, such as the Sybase IQ insert location command, to access net-change database instances. Each replicate table can have up to three tracking tables within a net-change database. You can inspect the net-change database and the tables within the database to monitor RTL replication and troubleshoot problems.

Use the sysadmin cdb command to monitor a net-change database. See "sysadmin cdb," in Chapter 3, "Replication Server Commands" in the *Replication Server Reference Manual*.

Mixed-version support and backward compatibility

RTL cannot replicate referential constraints specified in replication definitions if the outbound route version is earlier than 15.5.

RTL works if the inbound route version is earlier than 15.5. However, no referential constraint information is available to a Replication Server with version 15.5 or later.

Continuous replication is the default replication mode available to all supported versions of Replication Server. RTL is available only with Replication Server 15.5 and later.

Migrating from the staging solution to RTL

If you are using the staging solution to replicate to Sybase IQ, you can use this scenario to migrate to the real-time loading solution.

The scenario assumes a replication topology where ASE(pdb1) is the primary Adaptive Server database, PRS is the primary Replication Server, RRS is the replicate Replication Server, ASE(staging database) is the Adaptive Server staging database, and IQ(iqdb) is the replicate Sybase IQ database:

ASE(pdb) -->PRS --> RRS --> ASE(staging database) --> IQ(iqdb)

Before migrating

Before you migrate from the staging solution:

1. To use RTL, you must upgrade both the primary and replicate Replication Servers to version 15.5. See the *Replication Server Installation Guide* and *Replication Server Configuration Guide* for installation and upgrade instructions.
2. Verify that no transactions flow into pdb1 and that the replication system is quiesced during migration:
   a. Stop Replication Agent for all primary databases and system databases by executing on Replication Server:

      ```
      suspend log transfer from all
      ```

   b. Stop RepAgent for the RSSD if you are using Adaptive Server as the RSSD:

      ```
      sp_stop_rep_agent emb
      ```

   c. Verify that the Replication Server queues have been drained out and Replication Server has been quiesced by executing:

      ```
      admin quiesce_check
      ```

      Retry with admin quiesce_force_rsi if Replication Server is not quiesced yet. If Replication Server is not quiesced, data loss is possible.

   Verify that pdb1 and iqdb are synchronized. You can resynchronize the databases by loading data to iqdb from the staging database after all the data is replicated to the staging database. If you do not a resynchronize the databases, you must purge and materialize iqdb.
3. Add an entry for the replicate Replication Server (RRS) to the Sybase IQ interface file to allow the Sybase IQ server to connect to the replicate Replication Server and pull data.

Migrating to the real-time loading solution

1. Create a maintenance user in the iqdb replicate Sybase IQ data server or you can use the existing maintenance user.
2. Create the connection to the replicate Sybase IQ database from the replicate Replication Server using the rs_ase_to_iq connection profile and the maintenance user from step 2, such as dbmaint:

```
create connection to IQSRVR.iqdb
using profile rs_ase_to_iq;standard
set username to dbmaint
set password to dbmaint
go
```

3 At the primary database, if a table owned by dbo is not marked as owner_on, you must enable owner_on for the table so that Sybase IQ can find the table since dbo does not exist in Sybase IQ:

   - ASE 15.0.3 and later:

     ```
     sp_setrepdefmode testtab, 'owner_on'
     go
     ```

   - Versions earlier than ASE 15.0.3:

     ```
     sp_setreptable testtab, 'true', 'owner_on'
     go
     ```

4 Recreate the replication definition to include owner information since you have enabled owner_on.

5 If there are referential constraints between tables, you must alter the replication definition to define referential constraints so that Replication Server is aware of the referential constraints and can perform bulk apply in the proper order. See “Tables with referential constraints” on page 164

6 Enable RTL for the connection to the replicate database:

   ```
   alter connection to iqserver_name.rdb
   set dsi_compile_enable to 'on'
   ```

   After suspending and resuming the connection, the change in the connection takes effect.

7 Create subscriptions for each table. If the primary and replicate database are synchronized, include the without materialization clause in the subscription. Otherwise you must enable autocorrection during materialization.

You can now replicate from Adaptive Server directly to Sybase IQ.

Cleaning up after migration

After enabling and configuring replication using RTL, you can clean up the systems in the staging solution:

1 Drop subscriptions of the staging database.

2 Drop the replication definition that you are not using.

3 Drop connections to the staging database from the replicate Replication Server.

4 Terminate the environment for pulling data from the staging database to Sybase IQ.

PART 4

Additional Oracle Replication Topics

Chapters in this part describe Oracle specific tasks:

- Chapter 12, “Managing Heterogeneous Warm Standby for Oracle,” describes one way to create and manage a warm standby application using Replication Server for Oracle databases.
- Chapter 13, “Resynchronizing Oracle Replicate Databases,” describes the procedures to resynchronize the Oracle databases in your replication environment.

CHAPTER 12

Managing Heterogeneous Warm Standby for Oracle

This chapter describes one way to create and manage a warm standby application using Replication Server for an Oracle database.

Overview

A **warm standby application** is a pair of databases, one of which is a backup copy of the other. Client applications update the **active database**; Replication Server maintains the **standby database** as a copy of the active database.

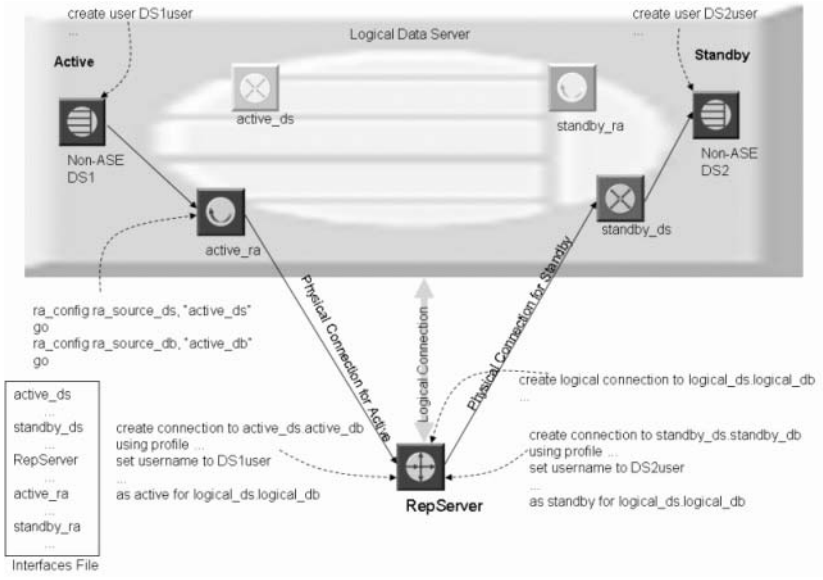
If the active database fails, or if you need to perform maintenance on the active database or on the data server, a switch to the standby database allows client applications to resume work with little interruption.

To keep the standby database consistent with the active database, Replication Server reproduces transaction information retrieved from the active database's transaction log. Subscriptions are not needed to replicate data into the standby database.

How a warm standby for Oracle works

Figure 12-1 illustrates the normal operation of an example warm standby application.

Figure 12-1: Warm standby application



In this warm standby application:

- Client applications execute transactions in the active database.
- The Replication Agent for the active database retrieves transactions from the transaction log and forwards them to Replication Server.
- Replication Server executes the transactions in the standby database.
- Replication Server may also copy transactions to destination databases and remote Replication Servers.

In a warm standby application, the active database and the standby database appear in the replication system as a connection from the Replication Server to a single logical database.

In many Replication Server applications:

- A primary database is the source of data that is copied to other databases through the use of replication definitions and subscriptions.
- A destination database receives data from the primary (source) database.

For detailed information about database connections, see Chapter 3, "Managing Warm Standby Applications" in the *Replication Server Administration Guide Volume 2*.

Warm standby requirements and restrictions

The following restrictions apply to all Replication Server warm standby applications:

- One Replication Server manages both the active and standby databases. Both the active and standby databases must be from the same vendor.
- Replication Server does not switch client applications to the standby database.
- Run the data server for the active and standby databases on different machines. Putting the active and standby databases on the same data server or hardware resources undermines the benefits of the warm standby feature.
- Sybase recommends that tables in the active and standby databases should have a primary key defined.

Function strings for maintaining standby database

Replication Server uses the system-provided function-string class rs_oracle_function_class for the standby DSI, which is the connection to the standby database.

- rs_marker – marks the point in the transaction log of the active database where replication must be enabled for the standby connection. Everything before the marker is not replicated while everything after the marker is replicated.
- rs_repl_off – disables replication for the current session.
- rs_triggers_reset – disables all triggers at the replicate database that triggers firing for the current session.
- rs_trunc_set – moves the truncation point used by the Replication Agent for Oracle to the end of the transaction log.

What information is replicated for an Oracle warm standby application?

Replication Agent supports different methods for enabling replication to the Oracle standby database. The level and type of information that Replication Server copies to the standby database depends on the method you choose; one of:

- pdb_setrepddl – allows replication of DDL commands and procedures that make changes to system tables stored in the database. You can use DDL commands to create, alter, and drop database objects, such as tables and views. Supported DDL system procedures affect information about database objects, and executed at the standby database by the DDL user.
- pdb_setreptable – marks all user tables or a specified table for replication.

Optionally, you can also use pdb_setrepproc to tell Replication Agent which user-stored procedures to replicate to the standby database.

For detailed information on Replication Agent for Oracle configuration parameters, see Chapter 2, “Configuration Parameters” in the *Replication Agent Reference Manual*.

Setting up warm standby databases

Setting up databases for a warm standby application involves three high-level tasks. Each of these tasks may include additional tasks.

1 Create a single logical connection to be used by both the active and standby databases.

2 Use the Replication Server create connection command to add the active database to the replication system. You need not add the active database if it has already been added to the replication system.

3 Use the Replication Server create connection command to add the standby database to the replication system.

Before you begin

Before setting up the databases:

- Install the Replication Server that manages the active and standby databases. A single Replication Server manages both the active and the standby database.
- Set up ECDA or ExpressConnect for Oracle connectivity. If using ECDA, one copy at primary site and one copy at standby site must be running and configured to communicate with the Oracle databases.
- Configure the Replication Agent, and verify that it is running in Admin mode for both the active and the standby database.
- Define the DDL user name in both the active and the standby database, and verify that it is configured in both the Replication Agents.The ddl_username parameter is the database user name included in LTL for replicating DDL commands to the standby or target database. This user must have permission to execute all replicated DDL commands at the target database. The DDL user must be different from the maintenance user. In addition, the DDL user must also have alter session permission to execute the DDL command as the user who executed at the active database. The ddl_password parameter is the password corresponding to the database user name.

Task one: Creating the logical connection

This section explains how to create the logical connection for the warm standby application.

Naming the logical connection

To name the logical connection, use the form *logical_ds.logical_db*.

There are two methods for naming the logical connection:

- If the active database has not yet been added to the replication system, use a different name for the logical connection than for the active database. Using unique names for the logical and physical connections makes switching the active database more straightforward.
- If the active database has previously been added to the replication system, use the *data_server* and *database* names of the active database for the logical connection name. The logical connection inherits any existing replication definitions and subscriptions that reference this physical database.

When you create a replication definition or subscription for a warm standby application, specify the logical connection instead of a physical connection. Specifying the logical connection allows Replication Server to reference the currently active database.

Creating the logical connection

1 Using a login name with sa permission, log in to the Replication Server that will manage the warm standby databases.

2 Execute the create logical connection command:

```
create logical connection to logical_ds.logical_db
```

The data server name and database name can be any valid object name. Typical values might include Oracle System Identifier (SID), to relate the logical connection to the physical Oracle implementation.

Task two: Adding the active database

This section explains how to reconfigure Replication Agent for the active database and how to add the active database to the replication system.

Initializing the Replication Agent

To configure Replication Agent for the active database, start the Replication Agent for Oracle (RAO) instance and connect to it using isql.

1 Set the archive log file path of the source Oracle database:

```
ra_config pdb_include_archives, true
go
ra_config pdb_archive_path, <path-to-oracle-
archive-directory>
go
```

2 Configure connection of Replication Agent to the primary database:

```
ra_config pds_host_name, <the host name of the
source oracle>
go
ra_config pds_port_number <the port number of the
source oracle>
go
ra_config pds_database_name,<the source oracle
```

```
database name>
go
ra_config  pds_username, <the oracle user for
Replication Agent>
go
ra_config pds_password, <password>
go
test_connection PDS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
PDS  succeeded
```

3 Configure the Replication Agent connection to Replication Server:

```
ra_config rs_host_name, <the host name of the
Replication Server>
go
ra_config rs_port_number, <the port number of the
Replication Server>
go
ra_config rs_username, <the Replication Server user
for Replication Agent>
go
ra_config rs_password, <password>
go
ra_config rs_source_ds ',' <the DCON server name>
go
ra_config rs_source_db ',' <the source oracle
database name>
go
```

Note If you are using ExpressConnect for Oracle, replace the DirectConnect sever name with the name of the Oracle instance. For example:

```
ra_config rs_source_ds, 'ordb'
go
rs_config rs_source_db, 'ordb'
go
```

---

4 Configuring the Replication Agent connection to ERSSD:

```
ra_config rssd_host_name <the host name of the
ERSSD>
go
ra_config rssd_port_number, <the port number of the
ERSSD>
go
ra_config rssd_username, <the ERSSD user for
Replication Agent>
go
ra_config rssd_password, <password>
go
ra_config rssd_database_name, <the database name of
the ERSSD>
go
test_connection RS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
RS  succeeded
```

5 If the character set of Replication Server is not the same as Replication Agent, update the Replication Server character set:

```
ra_config rs_charset, <the charset of the
Replication Server>
```

6 Create a replication definition for each table marked for replication:

```
ra_config pdb_auto_create_repdefs, true
go
```

7 Set automatic marking of user tables:

```
ra_config pdb_automark_tables, true
go
```

8 Initialize the Replication Agent transaction log:

```
pdb_xlog init
```

9 Enable DDL replication for the active database. Enter:

```
pdb_setrepddl enable
```

Note Some DDL commands are filtered even when DDL replication is enabled. If you are enabling DDL replication, you must also set ddl_password and ddl_username.

10 Create replication definitions for tables created before Replication Agent was initialized:

```
rs_create_repdef all
go
```

Note If you designate as the active database one that has already been added to the replication system, the Replication Agent for the active database is suspended when you create the logical connection.

Resume the Replication Agent:

```
resume
go
```

Creating connection to the active database

1 Log in to Replication Server using a login name with suitable permission.

2 Execute:

```
create connection to active_ds.active_db
using profile …
set username to …
set password to …
with log transfer on
as active for logical_ds.logical_db
```

If you are using ExpressConnect for Oracle, execute:

```
create connection to ordb.ordb/*oracle data server
name. database name*/
using profile rs_oracle_to_oracle;eco
set username to ...
set password to ...
with log transfer on
as active for logical_ds.logical_db
go
```

Alternatively, if you are using ECDA, execute:

```
create connection to dco2active.ordb/*dco instance
name.database name*/
using profile rs_oracle_to_oracle;ecda
set username to ...
set password to ...
with log transfer on
as active for logical_ds.logical_db
go
```

Task three: Adding the standby database

This section explains how to add the standby database to the replication system and prepare it for operation.

Initializing the standby database

Initialize the standby database with data from the active database using the dump and load technique. For detailed information about how to dump data from active database and load to standby database, refer the Oracle documentation.

Initializing the Replication Agent

To configure Replication Agent for the standby database, start the RAO instance and connect to it using isql.

1 Set the archive log file path of the standby Oracle databases:

```
ra_config pdb_include_archives, true
go
ra_config pdb_archive_path, <path-to-oracle-
archive-directory>
```

```
go
```

2 Configure connection of Replication Agent to the standby database:

```
ra_config pds_host_name, <the host name of the
standby oracle>
go
ra_config pds_port_number <the port number of the
standby oracle>
go
ra_config pds_database_name,<the standby oracle
database name>
go
ra_config  pds_username, <the oracle user for
Replication Agent>
go
ra_config pds_password, <password>
go
test_connection PDS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
PDS  succeeded
```

3 Configure the Replication Agent connection to Replication Server:

```
ra_config rs_host_name, <the host name of the
Replication Server>
go
ra_config rs_port_number, <the port number of the
Replication Server>
go
ra_config rs_username, <the Replication Server user
for Replication Agent>
go
ra_config rs_password, <password>
go
ra_config rs_source_ds ','<the DCON server name>
go
ra_config rs_source_db ','<the standby oracle
database name>
```

```
go
```

Note If you are using ExpressConnect for Oracle, replace the DirectConnect sever name with the name of the Oracle instance. For example:

```
ra_config rs_source_ds, 'ordb'
go
rs_config rs_source_db, 'ordb'
go
```

4 Configuring the Replication Agent connection to ERSSD:

```
ra_config rssd_host_name <the host name of the
ERSSD>
go
ra_config rssd_port_number, <the port number of the
ERSSD>
go
ra_config rssd_username, <the ERSSD user for
Replication Agent>
go
ra_config rssd_password, <password>
go
ra_config rssd_database_name, <the database name of
the ERSSD>
go
test_connection RS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
RS  succeeded
```

5 If the character set of Replication Server is not the same as Replication Agent, update the Replication Server character set:

```
ra_config rs_charset, <the charset of the
Replication Server>
```

6 Create a replication definition for each table marked for replication:

```
ra_config pdb_auto_create_repdefs, true
go
```

7 Set automatic marking of user tables:

```
ra_config pdb_automark_tables, true
go
```

8 Initialize the Replication Agent transaction log:

```
pdb_xlog init
```

9 Enable DDL replication for the active database:

```
pdb_setrepddl enable
```

Note Some DDL commands are filtered even when DDL replication is enabled. If you are enabling DDL replication, you must also set ddl_password and ddl_username.

10 Configure the Replication Agent to work in standby mode. Set the ra_standby configuration parameter to "true" to work in standby mode.

```
ra_config ra_standby,'true'
go
```

Creating connection to the standby database

1 Log in to Replication Server using a login name with suitable permission.

2 Execute:

```
create connection to standby_ds.standby_db
using profile …
set username to …
set password to …
with log transfer on
as standby for logical_ds.logical_db
```

Resuming connection to the active database and the standby database

During initialization of the standby database, Replication Server suspended the connection to the active database. Resume the connection by executing this command in the Replication Server:

```
resume connection to active_ds.active_db
```

After resuming connections to the active and standby databases, check the warm standby status:

```
admin logical_status [,logical_ds,logical_db]
```

```
go
```

Resuming the Replication Agents for the active and standby databases

Resume the Replication Agents for the active and standby databases to start scanning the database log for transactions to replicate. In each Replication Agent, enter:

```
resume
go
```

Switching the active and standby databases

Switch to the standby database when the active database fails, or you want to perform maintenance on the active database.

Before switching active and standby databases

Figure 12-2 illustrates a warm standby application for a database that does not participate in the replication system other than through the activities of the warm standby application itself. Figure 12-2 represents the warm standby application in normal operation, before you switch the active and standby databases

Figure 12-2: Warm standby application —before switching

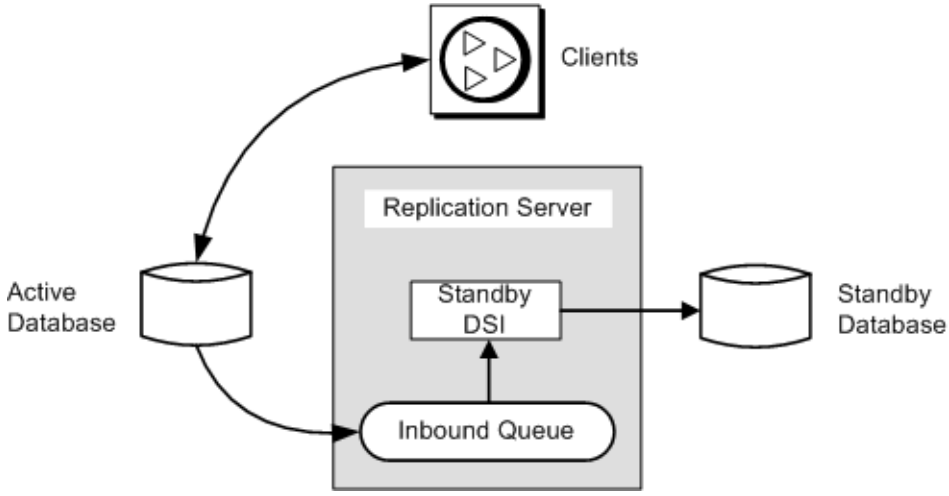


Figure 12-2 includes internal details to show that:

- Replication Server writes transactions received from the active database into an inbound message queue.
- This inbound queue is read by the DSI thread for the standby database, which executes the transactions in the standby database.

  Messages received from the active database cannot be truncated from the inbound queue until the standby DSI thread has read them and applied them to the standby database.

In this example, transactions are simply replicated from the active database into the standby database. The logical database itself does not:

- Contain primary data that is replicated to replicate databases or remote Replication Servers, or
- Receive replicated transactions from another Replication Server.

Internal switching steps

When you switch active and standby databases, Replication Server:

1 Issues suspend log transfer command against the active and standby connections.

2 Reads all messages left in the inbound queue and applies them to the standby database and, for subscription data or replicated stored procedures, to outbound queues.

Processes all committed transactions in the inbound queue before the switch completes.

3 Suspends the standby DSI.

4 Places a marker in the transaction log of the new active database. Replication Server uses this marker to determine which transactions to apply to the new standby database and to any replicate databases.

5 Stops the connection for the new active database. Purges the inbound queue and flushes the last oqid to rs_oqid table and resets rs_locator table accordingly. Resets the Replication Server segments flag and suspends the new standby DSI.

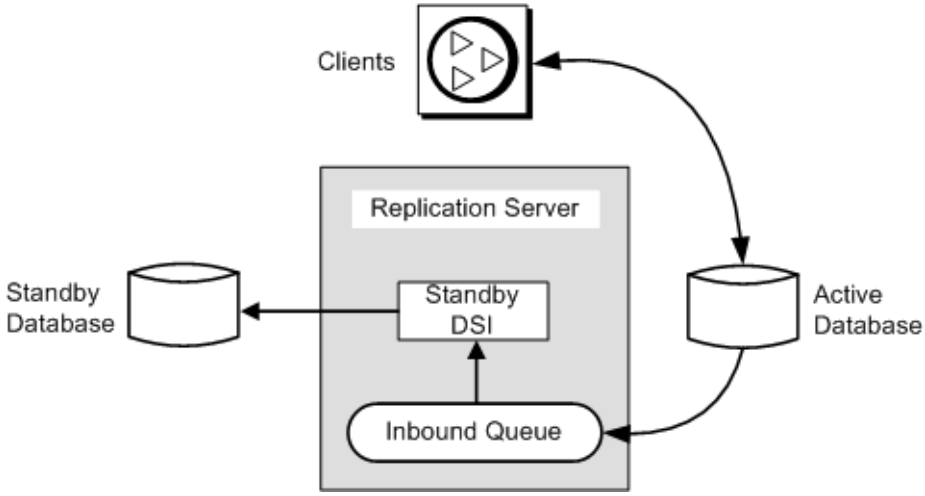
6 Updates the ptype parameter for both new active database and standby database. Replication Server marks subscriptions that are targeted for the old active database as valid and the subscriptions in the new active database as invalid.

7 Resumes the connection for the new active database, and resumes log transfer for the new active database so that new messages can be received. Resumes DSI for both the new active and standby databases.

After switching the active and standby databases

After you have switched the roles of the active and standby databases, the replication system changes, as shown in Figure 12-3.

Figure 12-3: Warm standby application—after switching



- The previous standby database is the new active database. Client applications have switched to the new active database.
- The previous active database, in this example, becomes the new standby database. Messages for the previous active database are queued for application to the new active database.

Switching the active and standby databases

1 At the Replication Server, enter:

```
switch active for logical_ds.logical_db
to standby_ds.standby_db
```

See “Internal switching steps” on page 188 for information on what Replication Server does when you switch.

2 To monitor the progress of a switch, enter:

```
admin logical_status, logical_ds, logical_db
```

The Operation in Progress and State of Operation in Progress output columns indicate the switch status.

3 When the active database switch is complete, resume the connection to the active database in Replication Server:

```
resume connection to active_ds.active_db
```

4 Suspend the Replication Agent at the original active site, if not already suspended. Configure it to standby mode:

```
ra_config ra_standby,true
```

5 Replication Agent at the original standby mode is automatically suspended and changed from standby mode to replication mode. To check, enter:

```
ra_config ra_standby
```

The return values must be false.

6 Resume both the Replication Agents at the active and standby sites.

Note If Replication Server stops in the middle of switching, the switch resumes after you restart Replication Server. When you resume the Replication Agent at the standby site, it automatically updates the Replication Agent System Database (RASD).

Monitoring a warm standby application

You can monitor warm standby applications using:

- Replication Server log file
- Using admin commands

Warm standby applications using replication

For warm standby applications that involve replication, the logical database serves as a primary or replicate database in the replication system.

For detailed information about warm standby applications using replication, see Chapter 3, “Managing Warm Standby Applications” in the *Replication Server Administration Guide Volume 2*.

Using replication definitions and subscriptions

For Oracle warm standby, the Replication Agent automatically creates the replication definition during initialization (execution of pdb_xlog init command) and by setting the configuration parameter pdb_auto_create_repdefs to true. When the Replication Agent detects a logical connection (by querying the RSSD), the replication definitions created by the Replication Agent are customized to support an Oracle warm standby environment.

In certain scenarios, where the warm standby environment is replicated to a non-warm standby database, you must create a second replication definition for each table or stored procedure you plan to replicate.

This section contains information about creating additional replication definitions when using warm standby databases.

Creating additional replication definitions for warm standby databases

For Oracle warm standby, the replication definitions are automatically created by the Replication Agent at initialization, and have these attributes:

- Mappings from Oracle datatypes to Replication Server user-defined datatypes (UDD) are provided.

- By default, tables with clob/blob columns are defined with the always_replicate clause. If auto_create_repdefs is set to “on”, then clob/blob columns are defined with replicate_if_change clause.

  Note The always_replicate and replicate_if_change are clauses for creating replication definitions.

- The replication definitions are created with the send standby replication definition columns clause.

If you are replicating from a warm standby primary environment to a replicate database outside the warm standby environment, you may want to create a new replication definition for each table to be replicated for these cases to:

- Provide mappings from Oracle datatypes to Replication Server UDD. See Appendix A, “Datatype Translation and Mapping” for Replication Server datatype names for Oracle.
- Use replicate_if_change clause for table with clob/blob column.
- Include the send standby all columns clause, if a database level subscription is used to subscribe to the non-warm standby database.
- Specify the primary and replicate function owner. Customize the function string to specify the target (standby database) function owner information for user procedures.

For example, if a user table TB1 is defined with one of the columns “COL5”as Oracle datatype date, to replicate the column to standby database as expected, the user must create a replication definition as shown:

```
create replication definition rep1
with primary at ordb.pdb
with all tables named 'USER1'.'TB1'
(
"COL1" int,
"COL2" int,
"COL3" int,
"COL4" char(255),
"COL5" rs_oracle_datetime,
)
primary key( "COL1","COL2","COL3")
searchable columns( "COL1","COL2","COL3","COL5")
send standby replication definition columns
replicate minimal columns
go
```

In this example, the clause send standby replication definition columns in create replication definition command specifies that this replication definition can be used for a subscribed database as well as for a standby database.

Using subscriptions with warm standby applications

Although subscriptions are not used in replicating from the active database to the standby database, you can:

- Create subscriptions for the data in a logical primary database, or
- Create subscriptions to replicate data from other databases into a logical replicate database.

The create subscription and define subscription commands use the logical database and data server names instead of the physical names.

For detailed information about warm standby applications using replication, see Chapter 3, “Managing Warm Standby Applications” in the *Replication Server Administration Guide Volume 2*.

Upgrade and downgrade considerations

Provides information related to Replication Server 15.5 upgrade or downgrade for the Oracle warm standby applications.

Upgrading

Oracle warm standby feature does not require any special instructions that you may need to perform after an upgrade. See Chapter 3, “Upgrading or Downgrading an Existing Replication Server” in the *Replication Server Configuration Guide* for your platform for information on upgrading your replication server version.

Downgrading

After you have completed downgrading the RSSD using the rs_init, your connection to Oracle data server (if your connection to Oracle is created using the create connection with using profile clause) may be down because the wait_after_commit configuration parameter provided in Replication Server 15.5 is no longer available. To resume your replication:

1 Execute:

```
alter connection to data_server.database
set dsi_serialization_method to 'wait_for_commit'
go
```

2 Resume log transfer to active database.

3 Resume Replication Agent for Oracle.

After you have completed performing these above steps, you can start replicating from active database to other replicate database with the replication definitions and subscriptions created before downgrade.

Note After downgrading, the Oracle standby connection will be broken as it is not able to do function string mapping or data translation. If you did not drop standby connection before downgrading the RSSD, you can drop it after the downgrade.

CHAPTER 13

Resynchronizing Oracle Replicate Databases

This chapter describes how to resynchronize the Oracle databases in your replication environment. The chapter discusses how resynchronication works and the components involved, and the provides some scenarios showing how to prepare, configure, and execute resynchronization.

Introduction

Replication Server allows you to resynchronize and materialize the replicate database, and resume further replication without loss or risk inconsistency of data, and without forcing a quiesce of your primary database.

Database resynchronization is based on obtaining a dump of data from a trusted source and applying the dump to the target database you want to resynchronize.

Database resynchronization requires a version of a Replication Agent for your database that supports this feature. For the specific commands for Replication Agent, see the Replication Agent documentation.

Product compatibility

Table 13-1 lists the versions of Oracle, Replication Agent for Oracle, ECDA Option for Oracle, and ExpressConnect for Oracle that support the resynchronization of Oracle databases. With Replication Server Options 15.5, ExpressConnect for Oracle replaces ECDA Option for Oracle.

See the Replication Server Options documentation and the *Replication Server Heterogeneous Replication Guide*.

Table 13-1: Product compatibility for resynchronizing Oracle databases

Database server version	Replication Agent version	ExpressConnect and ECDA for Oracle versions
Oracle Server 10g, 11g	15.5	ECDA 15.0 ESD #3, ExpressConnect 15.5 for Oracle

Configuring database resynchronization

To resynchronize Oracle databases:

1 Stop replication processing by suspending Replication Agent.

2 Place Replication Server in resync mode. While in resync mode, Replication Server skips transactions and purges replication data from replication queues in anticipation of the replicate database being repopulated from a dump taken from the primary database or trusted source.

3 Obtain a dump from the primary database.

4 Restart Replication Agent and send a resync database marker to Replication Server to indicate that a resynchronization effort is in progress.

   When Replication Server detects a dump marker that indicates the completion of the primary database dump, Replication Server stops skipping transactions and can determine which transactions to apply to the replicate database.

5 Apply the dump to the replicate database.

6 Resume replication.

See “Database resynchronization scenarios” on page 201 to use these steps in different user scenarios.

You must use commands and parameters from both Replication Server and Replication Agent for Oracle for database resynchronization.

Instructing Replication Server to skip transactions

Use the skip to resync parameter with the resume connection command to instruct Replication Server to skip transactions in the DSI outbound queue for the specified replicate database until Replication Server receives and acknowledges a dump database marker sent by Replication Agent. Replication Server skips processing of records in the outbound queue since the data in the replicate database is expected to be replaced with the dump contents. See "resume connection," in Chapter 3 "Replication Server Commands" in the *Replication Server Reference Manual*.

Warning! If you execute resume connection with the skip to resync marker option on the wrong connection, data on the replicate database becomes unsynchronized.

Syntax

resume connection to *data_server*.*database*
[skip [n] transaction | execute transaction | skip to resync marker]

When you set skip to resync marker, Replication Server does not log the transactions that are skipped in the Replication Server log or in the database exceptions log. Replication Server logs transactions that are skipped when you set skip [n] transaction.

Sending the resync database marker to Replication Server

You can configure or instruct Replication Agent for Oracle to send a resync database marker to Replication Server to indicate that a resynchronization effort is in progress. When you restart Replication Agent in resync mode, Replication Agent sends the resync database marker to Replication Server as the first message before Replication Agent sends any SQL data definition language (DDL) or data manipulation language (DML) transactions. Multiple replicate databases for the same primary database all receive the same resync marker since they each have a DSI outbound queue.

For each DSI that resumes with the skip to resync marker parameter, the DSI outbound queue records in the Replication Server system log that DSI has received the resync marker and also records that from that point forward, DSI rejects committed transactions until it receives the dump database marker.

In Replication Agent for Oracle, use resume with the resync or resync, init parameters to support these corresponding options for sending the resync database marker. See “resume,” in Chapter 1, “Command Reference” in the *Replication Agent Reference Manual*.

Sending a resync marker

Replication Agent can automatically determine whether the truncation point has changed. You can send a resync marker using resume resync when:

- There is no change to the truncation point and the expectation is that the Replication Agent should continue processing the transaction log from the last point that the Replication Agent processed. Each outbound DSI thread and queue receives and processes the resync database marker. DSI reports to the Replication Server system log when a resync marker has been received, satisfying the skip to resync marker request of DSI. Subsequently, DSI rejects committed transactions while it waits for a dump database marker. With this message and the change of behavior to one of waiting for the dump database marker, you can apply any dump to the replicate database at this time.
- The truncation point of the primary database has been moved in time. This can occur when you manually change the truncation point.

  In this situation, before sending the resync marker, execute ra_init force in Replication Agent, which reinitializes the Replication Agent repository. With this reinitialization, Replication Agent tracks any changes in the database that it might miss as a result of moving the truncation point and skipping the processing of some transaction log records.

  Since the truncation point has changed, open transactions in the Replication Server inbound queue must be purged because these transactions do not match new activity sent from the new truncation point. Replication Server resets duplicates checking, since the changed truncation point could send a record with a previous origin queue ID (OQID). Since the prior data is purged from the queues, Replication Server does not treat any new activity from the Replication Agent as duplicate activity, and consequently does not reject the new activity. The purge option does not change DSI processing because Replication Server continues to reject outbound queue commands while waiting for the marker.

Sending a resync marker with the *init* command

Send a resync marker with an init command using resume resync, init to instruct Replication Server to purge all open transactions from the inbound queue, reset duplicate detection, and suspend the outbound DSI. Use this option to reload the primary database from the same dump as the replicate database.

Since there is no dump taken from the primary database, Replication Agent does not send a dump database marker. Instead of waiting for a dump database marker after the resync marker, the init option suspends the DSI connection immediately after Replication Server processes the resync marker. After DSI is suspended, all subsequent activity through DSI consists of new transactions. You can resume DSI once you reload the replicate database from the same dump you used for the primary. See “Resynchronizing both the primary and replicate databases from the same dump” on page 205.

Obtaining a dump of the database

You can use any dump utility to obtain a dump of the database. When the dump is complete, you, as the administrator, must determine the desired dump point based on information obtained from the primary database when the dump was taken. The dump utility may provide the dump point. The scenarios in later sections use the Oracle RMAN utility.

To obtain the dump point from one RMAN backup set, use the list backup Oracle command. This is an example of the output from list backup:

```
List of Backup Sets
===================
BSKey   Type LV  Size       Device Type Elapsed Time Completion Time
------- ---- --  ---------- ----------- ------------ ---------------
1       Full     1.13G          DISK       00:02:26     19-MAY-09
BP Key: 1   Status: AVAILABLE  Compressed: NO  Tag: TAG20090519T102428
Piece Name: D:\ORACLE11G\PRODUCT\11.1.0\DB_1\DATABASE\01KFDJAS_1_1
List of Datafiles in backup set 1
File LV Type Ckp SCN    Ckp Time  Name
---- -- ---- ---------- --------- ----
1       Full 1721510    19-MAY-09 D:\ORACLE11G\ORADATA\ORAPDB11G\SYSTEM01.DBF
2       Full 1721510    19-MAY-09 D:\ORACLE11G\ORADATA\ORAPDB11G\SYSAUX01.DBF
3       Full 1721510    19-MAY-09 D:\ORACLE11G\ORADATA\ORAPDB11G\UNDOTBS01.DBF
4       Full 1721510     19-MAY-09 D:\ORACLE11G\ORADATA\ORAPDB11G\USERS01.DBF
```

The value of “Ckp SCN”, 1721510, is the required dump point.

See Oracle documentation for more information on the RMAN utility and the list backup command.

Sending the dump database marker to Replication Server

When the Data Server Interface thread (DSI) is in resync mode after you resume DSI using skip to resync marker, and you restart Replication Agent in resync mode, the dump database marker received after the resync database marker suspends DSI and removes any existing resynchronization state for that DSI connection.

Multiple replicate databases for the same primary database all receive the same dump database marker. In Replication Agent, the dump database marker is sent based on the Replication Agent configuration setting, using the lr_dump_marker command with the *oracle scn* parameter. See the Replication Agent documentation.

Note When you restart Replication Agent using resync with init, DSI suspends immediately after receiving the resync database marker. DSI does not wait for a dump marker before suspending.

You can manually resume DSI after you apply the dump to the replicate database. DSI no longer rejects committed transactions and all transactions that commit after the dump point, which is indicated by the dump database marker, are replicated.

Monitoring DSI thread information

Use the admin who command to provide information on DSI during database resynchronization:

State	Description
Skip Until Resync	DSI is resuming after you execute skip to resync. This state remains until DSI receives a resync database marker.
Skip Until Dump	DSI has received a resync database marker. This state remains until DSI has processed a subsequent dump database marker.

Applying the dump to a database to be resynchronized

You can load the primary database dump to the replicate database only after you see these messages in the system log:

- When Replication Server receives the resync database marker:

  ```
  DSI for data_server.database received and processed
  Resync Database Marker. Waiting for Dump Marker.
  ```

- When Replication Server receives the resync database with init marker:

  ```
  DSI for data_server.database received and processed
  Resync Database Marker. DSI is now suspended. Resume
  after database has been reloaded.
  ```

- When Replication Server receives the dump database marker:

  ```
  DSI for data_server.database received and processed
  Dump Marker. DSI is now suspended. Resume after
  database has been reloaded.
  ```

See Oracle documentation for instructions to load the dump to the database you want to resynchronize.

Database resynchronization scenarios

There are procedures you must follow to resynchronize databases in different scenarios. After completing these procedures, the primary and replicate databases are transactionally consistent.

To execute these procedures, you must:

- Be a replication system administrator
- Have an existing replication environment that is running successfully
- Have methods and processes available to copy data from the primary database to the replicate database

For commands and syntax for:

- Replication Agent for Oracle – see the *Replication Agent Reference Manual*.
- Replication Server – see the *Replication Server Reference Manual*.

Resynchronizing one or more replicate databases directly from a primary database

Use this procedure to resynchronize one or multiple replicate databases from a single primary database. This procedure, with minor variations, allows you to:

- Repopulate the replicate database when the replication latency between primary and replicate databases is such that to recover a database using replication is not feasible, and reporting based on the replicate data may not be practical because of the latency.
- Repopulate the replicate database with trusted data from the primary database.
- Coordinate resynchronization when the primary database is the source for multiple replicate databases.

❖ **Resynchronizing directly from a primary database**

1 Stop replication processing by Replication Agent. Do not alter the truncation point. In Replication Agent, execute:

```
suspend
```

2 Suspend the Replication Server DSI connection to the replicate database:

```
suspend connection to dataserver.database
```

3 Instruct Replication Server to remove data from the replicate database outbound queue and wait for a resync marker from the primary database Replication Agent:

```
resume connection to data_server.database skip to
resync marker
```

4 Obtain a dump of the primary database contents. If you use Oracle Recovery Manager (RMAN) tool, obtain the last System Change Number (SCN) of the RMAN backup using the Oracle list backup command. In Replication Agent, set this SCN as the value of lr_dump_marker:

```
lr_dump_marker oracle scn
```

5 Instruct Replication Agent to start in resync mode and send a resync marker to Replication Server:

- If the truncation point has not been moved from its original position, execute this command in Replication Agent:

```
resume resync
```

- If the truncation point has been moved from its original position, you must reinitialize the Replication Agent repository before you send the resync marker. In the Replication Agent, execute:

  ```
  ra_init force
  go
  resume resync
  go
  ```

6 In the Replication Server system log, verify that DSI has received and accepted the resync marker from Replication Agent by looking for this message:

  ```
  DSI for data_server.database received and processed
  Resync Database Marker. Waiting for Dump Marker.
  ```

  After DSI processes the resync marker for the replicate database, you can apply the dump to the replicate database.

  Note If you are resynchronizing multiple databases, verify that the DSI connection for each of the databases you want to resynchronize has accepted the resync marker.

7 Apply the dump of the primary database to the replicate database. See your database documentation.

8 Verify that Replication Server has processed the dump database marker by looking for this message in the Replication Server system log:

  ```
  DSI for data_server.database received and processed
  Dump Marker. DSI is now suspended. Resume after
  database has been reloaded.
  ```

  When Replication Server receives the dump marker, the DSI connection automatically suspends.

9 After you apply the dump to the replicate database, resume DSI using:

  ```
  resume connection to data_server.database
  ```

Resynchronizing using a third-party dump utility

Use this procedure to coordinate resynchronization after you dump the primary database using a third-party dump utility, such asa disk snapshot.

Third-party tools do not interact as closely with the primary database as native database dump utilities. If your third-party tool does not record anything in the primary database transaction log that Replication Agent can use to generate a dump database marker, generate your own dump database markers to complete the resynchronization process. See your third-party tool documentation.

❖ **Resynchronizing after using a third-party utility to create a dump**

1 Stop replication processing by Replication Agent. Do not alter the truncation point. In Replication Agent, execute:

```
suspend
```

2 Suspend the Replication Server DSI connection to the replicate database:

```
suspend connection to dataserver.database
```

3 Instruct Replication Server to remove data from the replicate database outbound queue and wait for a resync marker from the primary database Replication Agent:

```
resume connection to data_server.database skip to
resync marker
```

4 Obtain a dump of the primary database contents using the third-party dump utility.

5 Determine the dump point based on information from the primary database when you took the dump, or information from the third-party tool. With a third-party tool, you are responsible for determining the dump point. For example, if you are using a disk replication tool, you can temporarily halt activity at the primary database to eliminate transactions in progress from the disk snapshot, and then use the “end of transaction log” point as the dump database marker.

6 Execute the stored procedure on the primary database for Replication Agent to mark the end of the dump position that you obtained in step 5:

```
lr_dump_marker oracle scn
```

7 Instruct Replication Agent to start in resync mode and send a resync marker to Replication Server:

- If the truncation point has not been moved from its original position, execute this command in Replication Agent:

```
resume resync
```

- If the truncation point has been moved from its original position, you must reinitialize the Replication Agent repository before you send the resync marker. In the Replication Agent, execute:

  ```
  ra_init force
  go
  resume resync
  go
  ```

  Replication Agent automatically generates a dump database marker at a time based on the end of dump position that you obtained in step 5 and set in step 6, and sends the dump database marker to Replication Server.

8 Verify that DSI has received and accepted the resync marker from Replication Agent by looking for this message in the Replication Server system log:

  ```
  DSI for data_server.database received and processed
  Resync Database Marker. Waiting for Dump Marker.
  ```

9 Apply the dump of the primary database from the third-party tool to the replicate database. See your database and third-party tool documentation.

10 Verify that Replication Server has processed the dump database marker by looking for this message in the Replication Server system log:

  ```
  DSI for data_server.database received and processed
  Dump Marker. DSI is now suspended. Resume after
  database has been reloaded.
  ```

  When Replication Server receives the dump marker, the DSI connection automatically suspends.

11 After you apply the dump to the replicate database, resume DSI:

  ```
  resume connection to data_server.database
  ```

Resynchronizing both the primary and replicate databases from the same dump

Use this procedure to coordinate resynchronization to reload *both* the primary database and replicate database from the same dump or copy of data. No dump database marker is needed, since you are not obtaining a dump from the primary database.

❖ **Resynchronizing both primary and replicate databases from the same dump**

1 Stop replication processing by Replication Agent. Do not alter the truncation point. In Replication Agent, execute:

```
suspend
```

2 Suspend the Replication Server DSI connection to the replicate database:

```
suspend connection to data_server.database
```

3 Instruct Replication Server to remove data from the replicate database outbound queue and wait for a resync marker from the primary database Replication Agent:

```
resume connection to data_server.database skip to
resync marker
```

4 Apply the dump of the data from the external source to the primary database.

5 Move the truncation point to the end of the transaction log for the primary database. In Replication Agent, execute:

```
pdb_xlog move_truncpt
go
```

6 Reinitialize Replication Agent repository based on the latest system data from the primary database:

```
ra_init force
go
```

7 Instruct Replication Agent to start in resync mode with the init option. In Replication Agent, execute:.

```
resume resync, init
```

8 Verify that DSI has received and accepted the resync marker from the Replication Agent by looking for this message in the Replication Server system log:

```
DSI for data_server.database received and processed
Resync Database Marker. DSI is now suspended. Resume
after database has been reloaded.
```

When Replication Server receives and processes the resync database with init marker, the DSI connection suspends.

9 Apply the dump of the data from the external source to the replicate database.

10 After you apply the dump to the replicate database, resume DSI to the replicate database to allow Replication Server to apply transactions from the primary database:

```
resume connection to data_server.database
```

Resynchronizing the active and standby databases in a warm standby application

Use this procedure to resynchronize the active and standby databases in a warm standby environment, when the warm standby pair is the replicate site for a single primary database. In this scenario, the active, standby, and primary databases are Oracle databases.

❖ **Resynchronizing the active and standby databases when the warm standby environment is the replicate site**

1 Stop replication processing by both the primary database Replication Agent and the warm standby active database Replication Agent. Do not alter the truncation point. In Replication Agent, execute:

```
suspend
```

2 Suspend the Replication Server DSI connection to the active and standby databases:

```
suspend connection to dataserver.database
```

3 Instruct Replication Server to remove data from the outbound queue of the active and standby databases, and wait for a resync marker from the primary database Replication Agent:

```
resume connection to data_server.database skip to
resync marker
```

4 Obtain a dump of the primary database contents. See your database documentation. If you use Recovery Manager (RMAN) for Oracle, obtain the last System Change Number (SCN) of the RMAN backup. In the primary Replication Agent, configure this SCN as the value of lr_dump_marker :

```
lr_dump_marker oracle scn
```

5 Instruct the primary Replication Agent to start in resync mode and send a resync marker to Replication Server.

- If the truncation point in the primary database has not been moved from its original position, execute this command in the primary Replication Agent:

  ```
  resume resync
  ```

- If the truncation point in the primary database has been moved from its original position, reinitialize the Replication Agent repository before you send the resync marker. In the primary Replication Agent, execute:

  ```
  ra_init force
  go
  resume resync
  go
  ```

6 Verify that DSI for the active database has received and accepted the resync marker from the primary database Replication Agent by looking for this message in the Replication Server system log:

  ```
  DSI for data_server.database received and processed
  Resync Database Marker. Waiting for Dump Marker.
  ```

7 Verify that the Replication Server DSI for the active database has processed the dump database marker by looking for this message from the active database In the Replication Server system log:

  ```
  DSI for data_server.database received and processed
  Dump Marker. DSI is now suspended. Resume after
  database has been reloaded.
  ```

8 Apply the dump of the primary database to the active database. See your database documentation.

9 Move the truncation point to the end of the transaction log for the active database. In Replication Agent, execute:

  ```
  pdb_xlog move_truncpt
  go
  ```

10 Reinitialize Replication Agent repository based on the latest system data from the active database:

  ```
  ra_init force
  go
  ```

11 Instruct Replication Agent for the active database to start in resync mode with the init option. In Replication Agent, execute:.

  ```
  resume resync, init
  ```

12 Verify that DSI for the standby database has received and accepted the resync marker from the active database Replication Agent by looking for this message In the Replication Server system log:

```
DSI for data_server.database received and processed
Resync Database Marker. DSI is now suspended. Resume
after database has been reloaded.
```

When Replication Server receives and processes the resync database with init marker, the DSI connection suspends.

13 Obtain a dump of the active database contents and apply the dump to the standby database. You can also apply the dump of the primary database from step 4 if the dump does not include database configuration infomation.

14 Resume DSI to the active and standby databases:

```
resume connection to data_server.database
```

PART 5

Appendixes

Appendixes in this part provide supplemental information that can help you design and implement a replication system with heterogeneous or non-ASE data servers.

- Appendix A, “Datatype Translation and Mapping,” lists the class-level datatype translations for non-ASE data servers supported by Replication Server, and the Replication Server datatype names for non-Sybase datatypes.
- Appendix B, “Materialization Issues,” describes the issues related materializing subscriptions in a replication system with heterogeneous or non-ASE data servers.
- Appendix C, “Heterogeneous Database Reconciliation,” describes the issues related to synchronizing and reconciling databases in a replication system with heterogeneous or non-ASE data servers.
- Appendix D, “Troubleshooting Heterogeneous Replication Systems,” describes common problems and troubleshooting procedures for Sybase replication systems with heterogeneous or non-ASE data servers.
- Appendix E, “Reference Implementation for Oracle to Oracle Replication,” describes how you can quickly set up an Oracle to Oracle reference replication environment.

APPENDIX A

Datatype Translation and Mapping

This appendix lists the class-level datatype translations for non-ASE data servers supported by Replication Server, and the Replication Server datatype names for non-Sybase datatypes.

For each supported non-ASE data server, Replication Server provides class-level translations that define the default mapping from one datatype to another. Translations are provided for:

- Non-ASE datatypes that do not correspond directly to Adaptive Server datatypes
- Adaptive Server datatypes that do not correspond directly to the non-ASE datatypes
- Non-ASE datatypes that do not correspond directly to the datatypes of another supported non-ASE data server

Note Class-level translations are *not* provided for any datatype that corresponds directly to a datatype in another data server.

DB2 datatypes

This section lists class-level translations (default datatype mapping) for DB2 datatypes and Replication Server datatype names for DB2 datatypes.

Note This information about datatype translation applies to DB2 UDB in either mainframe environments (such as IBM z/OS), or UNIX and Microsoft Windows environments.

DB2 class-level translations

The following sections list the class-level translations for Adaptive Server datatypes to DB2 datatypes, DB2 datatypes to Adaptive Server datatypes, and DB2 datatypes to datatypes of other supported non-ASE data servers.

Adaptive Server to DB2 datatypes

Table A-1 lists class-level translations from Adaptive Server datatypes to DB2 datatypes.

Table A-1: Class-level translation from Adaptive Server to DB2 datatypes

Adaptive Server datatype	DB2 datatype
bigdatetime	TIMESTAMP
bigint	BIGINT
bigtime	TIMESTAMP
binary	CHAR FOR BIT DATA
bit	TINYINT
date	DATE (UNIX and Windows only)
datetime	TIMESTAMP
decimal	DECIMAL
int	NUMERIC
money	NUMERIC
numeric	NUMERIC
real	REAL (UNIX and Windows only)
smalldatetime	TIMESTAMP
smallint	NUMERIC
smallmoney	NUMERIC
time	TIME (UNIX and Windows only)
tinyint	NUMERIC
unsigned bigint	DECIMAL (20,0)

Adaptive Server datatype	DB2 datatype
unsigned int	BIGINT
unsigned smallint	INTEGER
unsigned tinyint	SMALLINT
unitext	DBCLOB
varbinary	VARCHAR FOR BIT DATA

DB2 to Adaptive Server datatypes

Table A-2 lists class-level translations from DB2 datatypes to Adaptive Server datatypes.

Table A-2: Class-level translation from DB2 to Adaptive Server datatypes

DB2 datatype	Adaptive Server datatype
CHAR FOR BIT DATA	binary
DATE	datetime
DECFLOAT UDB (UNIX and Windows only)	float
DOUBLE (UNIX and Windows only)	float
REAL (UNIX and Windows only)	real
TIME	datetime
TIMESTAMP	datetime
VARCHAR FOR BIT DATA	varbinary

DB2 to Microsoft SQL Server datatypes

Table A-3 lists class-level translations from DB2 datatypes to Microsoft SQL Server datatypes.

Table A-3: Class-level translation from DB2 to Microsoft SQL Server datatypes

DB2 datatype	Microsoft SQL Server datatype
CHAR FOR BIT DATA	binary
DATE	datetime
DECFLOAT UDB (UNIX and Windows only)	float
DOUBLE (UNIX and Windows only)	float
REAL (UNIX and Windows only)	real
TIME	datetime
TIMESTAMP	datetime
VARCHAR FOR BIT DATA	varbinary

DB2 to Oracle datatypes

Table A-4 lists class-level translations from DB2 datatypes to Oracle datatypes.

Table A-4: Class-level translation from DB2 to Oracle datatypes

DB2 datatype	Oracle datatype
CHAR FOR BIT DATA	RAW
DATE	DATE
DECFLOAT UDB (UNIX and Windows only)	FLOAT
DOUBLE (UNIX and Windows only)	FLOAT
REAL (UNIX and Windows only)	REAL
TIME	DATE (with time)
TIMESTAMP	DATE (with time)
VARCHAR FOR BIT DATA	RAW

Replication Server datatype names for DB2

Table A-5 lists the Replication Server user-defined datatype (UDD) names that identify DB2 datatypes for DB2 data servers on z/OS platforms.

Table A-5: Replication Server names for DB2 z/OS datatypes

DB2 z/OS datatype	Replication Server name
CHAR FOR BIT DATA	rs_db2_char_for_bit
DATE	rs_db2_date
DECIMAL	rs_db2_decimal, rs_db2_numeric
TIME	rs_db2_time
TIMESTAMP	rs_db2_timestamp
VARCHAR FOR BIT DATA	rs_db2_varchar_for_bit

Table A-6 lists the Replication Server UDD names that identify DB2 datatypes for DB2 data servers on UNIX and Microsoft Windows platforms.

Table A-6: Replication Server names for DB2 UNIX and Windows datatypes

DB2 UNIX and Windows datatypes	Replication Server name
CHAR FOR BIT DATA	rs_udb_char_for_bit
DATE	rs_udb_date
DECFLOAT	rs_udb_decfloat
DOUBLE	rs_udb_double
INTEGER	rs_udb_bigint
REAL	rs_udb_real
TIME	rs_udb_time
TIMESTAMP	rs_udb_timestamp
VARCHAR FOR BIT DATA	rs_udb_varchar_for_bit

Microsoft SQL Server datatypes

This section lists class-level translations (default datatype mapping) for Microsoft SQL Server datatypes and Replication Server datatype names for Microsoft SQL Server datatypes.

Microsoft SQL Server class-level translations

The following sections list class-level translations for Microsoft SQL Server datatypes to datatypes of other supported non-ASE data servers.

Adaptive Server to Microsoft SQL Server datatypes

Table A-7 lists class-level translations from Adaptive Server datatypes to Microsoft SQL Server datatypes for the unsigned datatypes.

The remaining class-level translations are not supplied for Adaptive Server datatypes to Microsoft SQL Server datatypes (or Microsoft SQL Server datatypes to Adaptive Server datatypes) because Microsoft SQL Server datatypes are directly compatible with Adaptive Server datatypes and they require no translation.

Table A-7: Class-level translation from Adaptive Server to Microsoft SQL Server datatypes

Adaptive Server datatype	Microsoft SQL Server datatype
unsigned bigint	DECIMAL (20,0)
unsigned int	BIGINT
unsigned smallint	INT
unsigned tinyint	SMALLINT
unitext	NTEXT

Microsoft SQL Server to DB2 datatypes

Table A-8 lists class-level translations from Microsoft SQL Server datatypes to DB2 datatypes.

Table A-8: Class-level translation from Microsoft SQL Server to DB2 datatypes

Microsoft SQL Server datatype	DB2 datatype
binary	CHAR FOR BIT DATA
bit	TINYINT
datetime	TIMESTAMP
decimal	DECIMAL
money	NUMERIC
numeric	NUMERIC
smalldatetime	TIMESTAMP
smallmoney	NUMERIC
varbinary	VARCHAR FOR BIT DATA

Microsoft SQL Server to Oracle datatypes

Table A-9 lists class-level translations from Microsoft SQL Server datatypes to Oracle datatypes.

Table A-9: Class-level translation from Microsoft SQL Server to Oracle datatypes

Microsoft SQL Server datatype	Oracle datatype
binary	RAW
datetime	DATE (with time)
money	DECIMAL
smalldatetime	DATE
smallmoney	DECIMAL
varbinary	RAW

Replication Server datatype names for Microsoft SQL Server

All Microsoft SQL Server datatypes are compatible with the corresponding Adaptive Server datatypes. Only one Microsoft SQL Server datatype has a user-defined datatype definition.

Table A-10 lists the Replication Server user-defined datatype (UDD) name that identifies a Microsoft SQL Server datatype.

Table A-10: Replication Server names for Microsoft SQL Server datatypes

Microsoft SQL Server datatype	Replication Server name
integer	rs_msss_bigint

Oracle datatypes

This section lists class-level translations (default datatype mapping) for Oracle datatypes and Replication Server datatype names for Oracle datatypes.

Oracle class-level translations

The following sections list class-level translations for Adaptive Server datatypes to Oracle datatypes, Oracle datatypes to Adaptive Server datatypes, and Oracle datatypes to datatypes of other supported non-ASE data servers.

Adaptive Server to Oracle datatypes

Table A-11 lists class-level translations from Adaptive Server datatypes to Oracle datatypes.

Table A-11: Class-level translation from Adaptive Server to Oracle datatypes

Adaptive Server datatype	Oracle datatype
bigdatetime	TIMESTAMP (9)
bigint	NUMBER
bigtime	TIMESTAMP (9)
binary	RAW
date	DATE
datetime	DATE (with time)
money	DECIMAL
smalldatetime	DATE

Adaptive Server datatype	Oracle datatype
smallmoney	DECIMAL
time	DATE (with time)
unsigned tinyint	SMALLINT
unsigned smallint	INTEGER
unsigned int	NUMBER
unsigned bigint	NUMBER
unitext	NCLOB
varbinary	RAW

Oracle to Adaptive Server Datatypes

Table A-12 lists class-level translations from Oracle datatypes to Adaptive Server datatypes.

Table A-12: Class-level translation from Oracle to Adaptive Server datatypes

Oracle datatype	Adaptive Server datatype
RAW	varbinary
DATE	datetime
TIMESTAMP (9)	datetime

Oracle to DB2 datatypes

Table A-13 lists class-level translations from Oracle datatypes to DB2 datatypes.

Table A-13: Class-level translation from Oracle to DB2 datatypes

Oracle datatype	DB2 datatype
RAW	CHAR FOR BIT DATA
DATE	DATE
DATE (with time)	TIMESTAMP
FLOAT	DOUBLE (UNIX and Windows only)
INTEGER	INTEGER (UNIX and Windows only)
TIMESTAMP (9)	TIMESTAMP (UNIX and Windows only)

Oracle to Microsoft SQL Server datatypes

Table A-14 lists class-level translations from Oracle datatypes to Microsoft SQL Server datatypes.

Table A-14: Class-level translation from Oracle to Microsoft SQL Server datatypes

Oracle datatype	Microsoft SQL Server datatype
RAW	varbinary
DATE	datetime
TIMESTAMP (9)	datetime

Replication Server datatype names for Oracle

Table A-15 lists the Replication Server user-defined datatype (UDD) names that identify Oracle datatypes.

Table A-15: Replication Server names for Oracle datatypes

Oracle datatype	Replication Server name
RAW	rs_oracle_binary
DATE	rs_oracle_datetime
ROWID	rs_oracle_rowid
INTEGER	rs_oracle_int
INTERVAL	rs_oracle_interval
BINARY_FLOAT	rs_oracle_float
NUMBER	rs_oracle_decimal
TIMESTAMP(n)	rs_oracle_timestamp9
TIMESTAMP(n) (with local time zone)	rs_oracle_timestamptz
UDD object type	opaque

APPENDIX B

Materialization Issues

This appendix describes the subscription materialization issues that you must consider when implementing a replication system with heterogeneous or non-ASE data servers. It also describes how to materialize subscriptions to primary tables in a non-ASE database.

Materialization overview

Materialization is creating and activating subscriptions and copying data from a primary database to a 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 the replicate objects (such as tables) are in a state consistent with those in the primary database.

Types of materialization

Replication Server supports two types of subscription materialization:

- *Bulk materialization* – 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* – creating a subscription and populating a replicate database using Replication Server commands.

For more information about subscription materialization methods, see the *Replication Server Administration Guide*.

Heterogeneous materialization issues

To materialize subscriptions to primary data in a non-ASE data server, you may use a bulk materialization or automatic materialization, if it applies. With bulk materialization methods, you must coordinate and manually perform the following activities:

- Define, activate, and validate the subscription (or create the subscription without materialization).
- Unload the subscription data at the primary database.
- Move the unloaded data to the replicate database site.
- Load the primary data into the replicate database tables.
- Resume the database connection from the replicate Replication Server to the replicate data server so that the replicate database can receive replicated transactions.
- Resume replication at the Replication Agent instance.

Bulk materialization options

There are two bulk materialization options for subscriptions to primary data in a non-ASE database:

- *Atomic bulk materialization*
  - Stop updates to the primary table and dump the subscription data from the primary database.
  - In the replicate Replication Server, define the subscription.
  - In the primary database, use the rs_marker function to activate the subscription using the with suspension option. See the *Replication Server Reference Manual* for information about applying the rs_marker function.
  - Load the subscription data into the replicate table.
  - Resume the database connection from the replicate Replication Server to the replicate database.
  - In the replicate Replication Server, validate the subscription.
- *Nonatomic bulk materialization*
  - In the replicate Replication Server, use the set autocorrection command.
  - In the replicate Replication Server, define the subscription.
  - In the primary database, use the rs_marker stored procedure to activate the subscription using the with suspension option.

- Dump the subscription data from the primary database.
- In the primary database, use the rs_marker stored procedure to validate the subscription.
- Load the subscription data into the replicate table.
- Resume the database connection from the replicate Replication Server to the replicate database.
- When the subscription becomes valid at all Replication Servers, turn off autocorrection.

Unloading data from a primary database

Part of the subscription materialization process involves unloading 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 data server software. You can use one of the OEM-supplied data unloading utilities or a database unload utility of your choice.

Note Once subscription data is unloaded from a primary database, you may need to perform datatype translation on the unloaded data before loading the data into the replicate database. See "Datatype translation issues."

Datatype translation issues

If you are not using the unload utility and are using automatic materialization, then Replication Server performs the translations.

If you use the heterogeneous datatype support (HDS) feature of Replication Server to perform either column- or class-level translations on replicated data, you must perform datatype translations on the subscription data you unload from the primary database for materialization.

Loading data into replicate databases

Part of the subscription materialization process involves loading subscription data from the primary table into the replicate table.

Note After subscription data is unloaded from a primary database, you may need to perform datatype translation on the unloaded data before loading the data into the replicate database.

If you are using Adaptive Server Enterprise as the data server for the replicate database, use the ASE bcp utility to load subscription data into the replicate database.

If you are using a non-ASE data server as the data server for the replicate database, you can use the load utility of your choice to load subscription data into the replicate database.

For more information about using bcp with Adaptive Server, see the *Adaptive Server Enterprise Utility Guide.*

Atomic bulk materialization

Atomic bulk materialization assumes that all applications updating the primary table can be suspended while a copy of the table is made. The copy is then loaded at the replicate site.

You can use this atomic bulk materialization to retrieve data from the primary database if you can (at least temporarily) suspend updates to the primary data.

Preparing for materialization

Before you start an atomic bulk materialization procedure, verify that:

- The primary table exists and contains data.
- You have access to 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 columns, datatypes.

- You have successfully configured all Replication Servers in your replication system.
- You have correctly created the replication definition at the primary Replication Server.
- If you are using Replication Agent for a DB2 UDB, Microsoft SQL Server, or Oracle primary database:
  - You have successfully initialized the Replication Agent which also creates some objects in the primary database.
  - You have 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.

Performing atomic bulk materialization

❖ **Performing atomic bulk materialization**

1 Use isql to log in to the replicate Replication Server as the system administrator (sa):

```
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 server name of the replicate Replication Server.

2 At the replicate Replication Server define the subscription:

```
1> define subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> [where search_conditions]
5> go
```

The dataserver.database must match the Replication Server connection name you use for the replicate database.

3 Check the subscription at both the primary and replicate Replication Servers. To verify that the subscription status is DEFINED, enter:

```
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 Perform any datatype translations necessary for the subscription data.

If any column-level translation is specified in the replication definition for this data, perform the datatype translation specified in the replication definition.

If class-level translations are specified for the subscription, perform the datatype translations specified for the subscription.

7 At the replicate Replication Server, activate the subscription:

```
1> activate subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> with suspension
5> go
```

8 Wait for the subscription to become active at both the primary and replicate Replication Servers. Execute check subscription 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.

9 Restore the primary table to read-write access (unlock).

10 Use the bcp or your site's preferred database utility to load the subscription data into the replicate database.

11 From the replicate Replication Server, resume the database connection for the replicate database:

```
1> resume connection
2> to dataserver.database
3> go
```

12 Validate the subscription at the replicate Replication Server:

```
1> validate subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

13 Wait for the subscription to become valid at both the primary and replicate Replication Servers, then execute check subscription 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 8, "DB2 UDB Replicate Data Server Issues for UNIX, Windows, and Linux."

See also

- *Replication Server Reference Manual* for information on Replication Command Language (RCL) commands
- *Replication Server Administration Guide* for information on configuring Replication Servers and materialization methods

Nonatomic bulk materialization

Nonatomic bulk materialization assumes applications updating the primary table cannot be suspended while a copy of the table is made. Therefore, 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 that:

- The primary table exists and contains data.

- You have access to 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 columns.
- You have successfully configured all Replication Servers in your replication system.
- You created the replication definition correctly at the primary Replication Server.
- If you are using Replication Agent for a DB2 UDB, Microsoft SQL Server, or Oracle primary database:
  - You have successfully initialized the Replication Agent which also creates some objects in the primary database.
  - You have 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.

Performing nonatomic bulk materialization

❖ **Performing nonatomic bulk materialization**

1 Use isql to log in to the replicate Replication Server as the system administrator (sa):

```
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.
- *RRS_servername* is the server name of the replicate Replication Server.

2 At the replicate Replication Server, turn on the autocorrection feature:

```
1> set autocorrection on
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

3 At the replicate Replication Server, define the subscription using the with suspension option:

```
1> define subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> with suspension
5> go
```

The dataserver.database must match the Replication Server connection name you use for the 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. 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 Perform any datatype translations necessary for the subscription data.

If any column-level translation is specified in the replication definition for this data, perform the datatype translation specified in the replication definition.

If class-level translations are specified for the subscription, perform the datatype translations specified for the subscription.

8 In the primary database, invoke the rs_marker stored procedure to validate the subscription.

9 Wait for the subscription to become valid at both the primary and replicate Replication Servers, then execute check subscription at both the primary and replicate Replication Servers to verify that the status is VALID.

10 Use the bcp utility or your site's preferred database load utility to load the subscription data into the replicate database.

11 From the replicate Replication Server, resume the database connection for the replicate database:

```
1> resume connection
2> to dataserver.database
3> 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 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:

```
1> set autocorrection off
2> for replication_definition
3> with replicate at dataserver.database
4> go
```

13 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 8, "DB2 UDB Replicate Data Server Issues for UNIX, Windows, and Linux."

See also

- *Replication Server Reference Manual* for information on Replication Command Language (RCL) commands
- *Replication Server Administration Guide* for information on configuring Replication Servers and materialization methods

Autocorrection

Replication Server can set autocorrection on a replication definition to prevent failures that would otherwise be caused by duplicate rows in a replicated table. The set autocorrection command corrects discrepancies that may occur during the nonatomic materialization process by converting each update or insert operation into a delete followed by an insert.

When you set autocorrection on a marked table using the Replication Agents API, ra_set_autocorrection, Replication Agent sends all columns, instead of sending only those columns that have changed in the update statement, to Replication Server. When Replication Agent sets autocorrection for one specific marked table or for all tables, the corresponding change applies to the primary database.

The primary databases that support autocorrection are:

- MS SQL Server
- IBM DB2
- Oracle – the autocorrection feature cannot work on LOB, LONG, LONG RAW, and user-defined type columns because of Oracle limitations for redo log recording.
- ASE

See the *Replication Agent 15.5 Primary Database Guide* for details on how to set autocorrection and the *Replication Agent 15.5 Reference Manual* for information on the ra_set_autocorrection command.

APPENDIX C

Heterogeneous Database Reconciliation

This appendix describes the issues involved with comparing and reconciling data from different databases in a heterogeneous replication system.

Sybase rs_subcmp utility

The Sybase rs_subcmp utility allows you to compare primary and replicate tables in Adaptive Server databases, and reconcile any differences. Sybase provides the rs_subcmp executable program with Replication Server.

Some other database vendors may provide a similar "compare" utility that can perform the same function for their own databases, but there is no equivalent utility to support different types of non-ASE data servers (for example, to compare tables in an Oracle database to tables in a Microsoft SQL Server database).

For non-ASE database support, you can either acquire third-party tools that provide such functionality, or build your own application.

Database comparison application

You can develop a custom application to perform the same functions as the rs_subcmp utility. The application's complexity depends on the number of

different data server types, the complexity of the tables to be compared, the amount of data translation involved, and so forth.

The following list describes the major issues that a database comparison application must accommodate to be successful in a heterogeneous replication environment:

- Connectivity – the application must be able to communicate with both the primary and replicate databases. If multiple database vendors are involved, ODBC and JDBC protocols can provide a common interface and functionality.
- Sort order – the default sort order may be different for different databases. The application may need to force the sort order to improve comparison performance.
- Character sets – some primary and replicate databases may store character data in different character sets. Your custom application may need to support these translations.
- Object identification – primary and replicate tables may not have identical names or exactly the same schema or column names. The comparison application may need to accept very explicit instructions for location, database, and table and column names to be referenced.
- Subset comparison – the application may need to compare only a portion of a table. The ability to specify a where clause type of select for both primary and replicate tables may be important.
- Latency – in a replication system, there is always some latency (a measure of the time it takes a primary transaction to appear in a replicate table). A comparison application must include some tolerance to distinguish between rows that are “not there” and “not there yet.”
- Data transformation – the application must be able to handle differences in precision and format between different databases, the same way Replication Server supports class-level translations. To simplify processing you want to allow certain columns to be excluded from the comparison process, based on datatype (for example, do not compare the DATE datatypes of different database vendors).
- Large object (LOB) data – large object (for example, LOB, CLOB, TEXT, or IMAGE) datatypes cause additional processing issues because of their size. To improve performance, limit the number of bytes used for comparison, if the likelihood of a “non-match” can still be relied on.

For more information about the rs_subcmp utility, see the *Replication Server Administration Guide* and the *Replication Server Reference Manual*.

APPENDIX D

Troubleshooting Heterogeneous Replication Systems

This appendix describes common problems and troubleshooting procedures for Sybase replication systems with heterogeneous or non-ASE data servers.

Troubleshooting overview

Common Replication Server troubleshooting tasks, such as dumping stable queues, debugging failures with the Data Server Interface (DSI) and Replication Server Interface (RSI), and diagnosing and correcting problems with subscriptions, are described in the *Replication Server Troubleshooting Guide*.

For non-ASE primary and replicate databases, the Replication Agent and ECDA gateway documentation provide troubleshooting information for each specific database.

This chapter describes some basic troubleshooting if replication fails in a Sybase replication system with heterogeneous or non-ASE data servers.

Inbound and outbound queue problems

The *inbound queue* is where Replication Server stores the data it receives from a primary database (through a Replication Agent or another Replication Server). The *outbound queue* is where Replication Server stores the data it needs to send to a replicate site (either a replicate database or another Replication Server).

This section describes how to troubleshoot problems with these Replication Server queues.

Inbound queue problems

You can tell that the Replication Server inbound queue for a primary database is not being updated if you issue the Replication Server admin who,sqm command at the primary Replication Server and the results indicate that:

- The number of blocks being written in the Replication Server inbound queue for the connection in question is not changing.
- The number of duplicate messages being detected is not increasing.

❖ **Determining the reason the inbound queue is not being updated**

1 Verify the Replication Server connection Replication Agent User thread status.

You can issue an admin who command in the primary Replication Server to review the status of the Replication Agent User thread for the Replication Server database connection in question.

- If there is no Replication Agent User thread for the connection, the connection was not created with the with log transfer on clause. You can alter the Replication Server database connection to turn log transfer processing on, if needed.
- If the Replication Agent User thread status is down, the Replication Agent is not actively connected to the Replication Server. A down status is typical for Replication Agents that connect to Replication Server only when there is work to be sent, and then disconnect after a period of inactivity.

2 Verify that the expected Replication Agent is executing.

Verify that the expected Replication Agent is active, and that the values of the Replication Agent rs_source_ds and rs_source_db configuration parameters match the desired Replication Server connection name.

Refer to the appropriate Replication Agent documentation for other tests to validate that the Replication Agent is executing.

3 Verify that the expected table or procedure is marked for replication.

Replication Agent documentation describes the Replication Agent commands you can use to check replication status.

Replication Agent provides for separate enabling of replication, in addition to marking. In this case, make sure the marked object is also enabled for replication.

4 Verify that the Replication Agent is scanning new records.

If the database object is marked for replication, the log scanning process of the Replication Agent should record that additional information is being scanned.

To verify that new records are being scanned:

- Start tracing in the Replication Agent.
- Update or execute a primary database object that has been marked for replication.
- Verify that scanning occurs.

Refer to the appropriate Replication Agent documentation to determine the trace flags you can use to validate the scanning process.

Outbound queue problems

You can tell that the Replication Server outbound queue for a replicate database is not being updated if you issue the Replication Server admin who,sqm command at the replicate Replication Server and the results indicate that:

- The number of blocks being written in the Replication Server outbound queue for the connection in question is not changing.
- The number of duplicate messages being detected is not increasing.

Problems between inbound and outbound queues are often naming problems.

❖ **Determining the reason the outbound queue is not being updated**

1 Verify that any Replication Server routes are active.

Refer to the Replication Server *Troubleshooting Guide* for route validation techniques between primary and replicate Replication Servers.

2 Verify that the Replication Server connection DSI thread is not down.

Issue an admin who command in the replicate Replication Server to review the status of the DSI thread for the Replication Server connection.

If the DSI thread status is down, the Replication Server is not connected to the replicate database (or ECDA gateway). Review the Replication Server log for errors and attempt to resume the connection.

3 Verify that the DSI thread connection is not in "Loss Detected" mode by viewing the replicate Replication Server log for "Loss Detected" messages for the DSI thread in question.

When Replication Server detects a loss, no further messages are accepted on the DSI thread connection.

Refer to the *Replication Server Administration Guide* for information about recovering from this error.

4 Verify the primary replication definition.

The primary Replication Server inbound queue can receive data, but when it cannot apply that data to any replication definition, the reason is that the name of the replication definition does not match the name presented in the Log Transfer Language (LTL) that was created by the Replication Agent. This becomes more likely when you are using different non-Sybase database types with different default character cases.

Replication Server processing of replication commands is case sensitive. In a replication system with non-ASE data servers, ensure that the LTL generated by Replication Agents matches the Replication Server connection names and replication definition object names.

Some Replication Agents always use lowercase names when they communicate with Replication Server (for example, Adaptive Server and DB2 UDB). However, the best option is to pick one character case (uppercase or lowercase) and use it consistently with all Replication Server connections, replication definitions, and subscription names.

Validating case-sensitivity is manual. You can use the rs_helprep command to verify the name of a replication definition. Then, you can then turn on LTL tracing in the Replication Agent and verify that the name provided in the LTL trace matches the spelling and character case of the name specified in the replication definition.

If the character case appears to be incorrect, review the Replication Agent documentation to verify the default character case settings and any possible configuration changes. If a name is misspelled, delete and then re-create the replication definition.

Replicate database is not updated

If the Replication Server outbound queue is being updated but transaction data is not being applied at the replicate database, use the following procedure to determine the reason.

❖ **Determining why replicate transactions are not applied at the replicate database**

1 Determine if the subscription contains a where clause.

Verify that the transaction data expected passes any where clause in the subscription definition. Use the rs_helpsub stored procedure to list the text of the subscription.

2 Verify HDS installation.

If you are using Replication Server HDS to support replication to or from a non-ASE data server, verify that the HDS connection profiles have been properly applied.

See “Expected datatype translations do not occur” on page 249.

3 Verify that the rs_lastcommit table is set up correctly.

If you are using Replication Server HDS to support replication to or from a non-ASE data server, verify that the HDS connection profiles have been properly applied.

Refer to “Updates to rs_lastcommit fail” on page 248.

4 Review the replicate Replication Server log for errors.

5 Review the replicate database log for errors.

6 Verify manual access to replicate objects.

Log in to the replicate database (or ECDA gateway) using the Replication Server connection maintenance user ID, and verify that you have update authority to the replicate table or procedure.

7 Validate commands sent to the replicate database:

- Turn on the DSI_BUF_DUMP trace flag in the replicate Replication Server and record to the Replication Server log the commands being sent to the replicate database.
- Verify that these commands, when manually applied, produce the expected results.

Note You can use the DSI_BUF_DUMP trace flag with any Replication Server. By contrast, the similar DSI_CMD_DUMP trace flag is available only with the diagnostic version of Replication Server. Refer to the *Replication Server Troubleshooting Guide* for more information about Replication Server traces.

8 Turn on tracing at the ECDA gateway to see what commands are being received.

For example, these parameters in the ECDA Option for Oracle configuration file cause ECDA to write additional information to the *DCO.log* file:

- network_tracing = 1
- traces = 1,2,3,4,5,6,10

Refer to the appropriate ECDA documentation for specific trace availability and syntax.

HDS issues and limitations

This section describes some known issues and limitations with the HDS feature in Replication Server.

Source value exceeds target datatype bounds

The datatype translations provided by Sybase specify that the thread attempting a translation where the source value exceeds the bounds of the target datatype should be stopped with the following error message:

```
E. 2007/12/14 11:14:54. ERROR #32055 DSI EXEC(135(1)
   snickers_dco.ora805) -
   /nrm/nrm.c(7023)
   Class Level translation for column/parameter
   'datetimecol' failed.
   Source DTID is 'datetime'.
   Target DTID is 'rs_oracle_datetime'.
   Function String Class ID 'rs_oracle_function_class'.
   Value length is '21'; Maximum target length is '20';
   The value is '99991231 23:59:59:010'
```

Typically, these are the most difficult translation problems to diagnose because there appears to be no problem with either the pairing of source/target datatypes or the value to be translated.

To diagnose this type of problem, you must be familiar with the datatype value boundary limits of all the translated target datatypes. For example, to diagnose the error shown, you must know that the upper boundary of an Oracle DATE value is 12/31/9999.

There are other options for datatype translations:

- Use the maximum value of the datatype definition.
- Use the minimum value for the datatype definition.
- Use the default value for the datatype definition.

Exact numeric datatype issues

There may be problems with exact numeric datatypes when the values replicated are at the boundaries (maximum or minimum values) of what is supported by the datatype definitions.

Microsoft SQL Server supports either 28 or 38 digits of precision, depending on how the server is started. By default, Microsoft SQL Server supports 28 digits of precision.

Sybase does not provide datatype definitions that support the Microsoft default of 28 digits of precision. Datatype definitions are not needed to support 38 digits of precision, because the Replication Server native numeric datatypes support up to 72 digits of precision.

When a number exceeds numeric precision of the Microsoft SQL Server replicate database, Replication Server returns the following error:

```
E. 2007/12/14 11:14:58. ERROR #1028 DSI EXEC(134(1)
   dcm_gabeat70_devdb.devdb)
   - dsiqmint.c(2888)
   Message from server: Message: 30291, State 0,
   Severity 19 --
   '[VENDORLIB] Vendor Library Error: [[Message
   Iteration=1|Data Source
   Name=mssql70_devdb|SQLState=22003|Native
   Error=1007|Message=[Microsoft][ODBC SQL Server
   Driver][SQL Server]The number
   '999999999999999999.999999999999999999' is out of
   the range for numeric representation (maximum
   precision 28).[Message Iteration=2|SQLState=22003|
   Native Error=|Message=[Microsoft][ODBC SQL Server
   Driver][SQL Server]The number
   '0.99999999999999999999999999999999999999' is out of
   the range for numeric representation (maximum
   precision 28).] <DCA>'
```

The most difficult numeric datatype issues involve precision and scale. Replication Server does not allow the precision and scale of a decimal datatype to be specified. A datatype definition can specify the maximum precision and maximum scale to be supported. However, if this does not equate to the specified precision and scale of an individual replicate column, then as the data approaches values near or at the boundaries, you may encounter problems that are reported differently, depending on the replicate data server.

For example, suppose you have a primary column declared as decimal (8,5) (8 digits of precision and a scale of 5), and suppose the replicate column is declared as decimal (6,4), even though the replicate data server can support a maximum of 7 digits precision and a scale of 7. In the replication definition, you specify the translation for the primary data server decimal datatype and for which there is a class-level translation to the replicate data server decimal datatype. Both datatype definitions specify the associated data servers maximum precision and scale.

If the value 999.99999 comes from the primary database, and the replicate data server's datatype definition specifies that rounding should be attempted, Replication Server attempts to apply a value of 1000.000. Even though this value satisfies the replicate database requirements for maximum precision and scale, it fails the precision and scale specified for this particular column. And if you specify for the replicate database's datatype definition that it should replace the value with the specified maximum value for the datatype definition, Replication Server attempts to apply a value of 9999999, which also fails the specified precision and scale for this particular column.

Error messages you might see from various data servers in this case include:

- The following DB2 error:

```
E. 2007/12/14 15:03:11. ERROR #1028 DSI EXEC(129(1)
   dwm5_via_rct.dwmdbas)
   - dsiqmint.c(2888)
   Message from server: Message: 30291, State 0,
   Severity 19 --
   '[VENDORLIB] Vendor Library Error: [[Message
   Iteration=1|SQLState=22003|Native Error=
   -413|Message=[Sybase][ClearConnect ODBC][DB2]The
   decimal or numeric value had an incorrect wire
   length compared to its specified FDOCA length
   10000000000000000000.00000000000] <DCA>'.
```

- The following Microsoft SQL Server error:

```
E. 2007/12/14 12:29:16. ERROR #1028 DSI EXEC(134(1)
   dcm_gabeat70_devdb.devdb)
   - dsiqmint.c(2888)
   Message from server: Message: 30291, State 0,
   Severity 19 --
   '[VENDORLIB] Vendor Library Error: [[Message
   Iteration=1|Data Source Name=mssql70_devdb|SQL
   Function=INSERT|SQLState=22003|Native Error=
   8115|Message=[Microsoft][ODBC SQL Server Driver]
   [SQL Server]Arithmetic overflow error converting
   numeric to data type numeric.[Message Iteration=
   2|SQLState=01000|Native Error=|Message=
   [Microsoft][SQL Server]The statement has been
   terminated.] <DCA>'
```

Numeric translation and identity columns in Microsoft SQL Server

Replication Server function strings to set identity insert off and on work in Microsoft SQL Server because it supports identity columns in the same manner as Adaptive Server. However, to support 28-digit precision in a Microsoft SQL Server database, the numeric datatype must be translated to the rs_msss_numeric datatype, and as a result, the identity characteristic is lost. To avoid this problem, the Microsoft SQL Server replicate table must not declare a translated numeric column as an identity.

If you attempt to replicate a translated numeric datatype into an identity column in Microsoft SQL Server, you receive an error similar to this:

```
E. 2007/12/14 12:05:39. ERROR #1028 DSI EXEC(134(1)
   dcm_gabeat70_devdb.devdb)
   - dsiqmint.c(2888)
   Message from server: Message: 30291, State 0,
   Severity 19 --
   '[VENDORLIB] Vendor Library Error: [[Message
   Iteration=1 |Data Source Name=mssql70_devdb|SQL
   Function=INSERT|SQLState=23000|Native Error=544
   |Message=[Microsoft][ODBC SQL Server Driver][SQL
   Server]Cannot insert explicit value for identity
   column in table 'ase_alltypes' when IDENTITY_INSERT
   is set to OFF.] <DCA>'
```

Troubleshooting specific errors

This section describes troubleshooting for specific errors you may encounter in a Sybase replication system with heterogeneous or non-ASE data servers.

Updates to rs_lastcommit fail

When replicating into a non-ASE replicate database, the replicate Replication Server updates the rs_lastcommit table as soon as the connection is resumed. If the replicate Replication Server error log displays a syntax error while updating the rs_lastcommit table, the following procedure may help identify the problem.

❖ **Troubleshooting rs_lastcommit update failure**

1 Verify that the table exists in the replicate database.

2 Verify access authority.

Log in to the replicate database using the Replication Server maintenance user ID and password specified in the create connection command for that database connection.

Verify that this user ID can update the rs_lastcommit table – you should be able to insert and delete a dummy entry without error.

3 Trace the actual command.

Turn on tracing in the replicate Replication Server (DSI_BUF_DUMP trace) or in the ECDA gateway and resume the Replication Server connection.

Identify the failing statement and correct as necessary.

Expected datatype translations do not occur

The most common reason for a datatype translation failure is an incomplete installation of the necessary user-defined datatypes (UDDs) and translations.

❖ **Validating UDD and translation installation**

1 Restart the Replication Servers. Replication Server caches all function-string information at start-up.

Subsequent changes to the function strings stored in the RSSD do not take effect until the Replication Server is restarted.

2 Verify that class-level translations have been applied to the replicate Replication Server.

The Replication Server connection profile provides the SQL statements necessary to apply class-level translations to the RSSD of the replicate Replication Server for a specific combination of non-ASE primary databases to non-ASE replicate databases.

Note The connection profile is required for any non-ASE replicate database. For example, if you are replicating from ASE to Oracle, the *rs_ase_to_oracle* connection profile for translations must be applied to ensure Replication Server updates to the rs_lastcommit table are properly translated and applied to the replicate database.

You can re-run these connection profiles without failure. Verify that your copy of the connection profiles has been updated with the correct use statement for the database name of the RSSD.

3 Verify that your replicate database Replication Server connection is associated with the appropriate function-string class.

To take advantage of class-level translations, the replicate Replication Server connection must use the correct non-ASE function-string class.

You can use the Replication Server rs_helpdb command to determine which function-string class is defined for a database connection.

Function-string classes for replicate databases are:

- Adaptive Server Enterprise – rs_sqlserver_function_class
- DB2 UDB on IBM z/OS platforms – rs_db2_function_class
- DB2 UDB on UNIX and Windows platforms – rs_udb_function_class
- Microsoft SQL Server – rs_msss_function_class
- Oracle – rs_oracle_function_class
- Sybase IQ – rs_iq_function_class

Use the Replication Server admin show_function_classes command to display a list of active function-string classes.

Use the Replication Server alter connection command to change the function-string class of an existing database connection.

4 Verify that the non-ASE function-string classes have been updated with appropriate function strings.

Replication Server connection profile *rs_xxx_xxx* provides the SQL statements necessary to apply function strings to the RSSD of the replicate Replication Server for a specific non-ASE replicate database.

For each function string, the connection profile issue a delete followed by an insert. You can re-run these connection profiles without failure.

Verify that your copy of the connection profile has been updated with the correct use statement for the database name of the RSSD.

5 Use the Replication Server admin translate command.

The admin translate command allows you to verify the results of a specific translation. Use this command to verify that the translation engine is providing the translation results you expect.

For more information about heterogeneous datatype support (HDS) and the admin translate command, refer to the *Replication Server Administration Guide*.

LTL generation and tracing

This section describes how to trace the Log Transfer Language (LTL) commands sent to a primary Replication Server, as well as other significant Replication Agent traces.

Replication Agent for DB2 UDB for z/OS

You can use the configuration parameters described in this section to obtain additional information that is not normally presented by Replication Agent for DB2 UDB for z/OS.

To print the log record identifier for each log record, and additional messages received from the DB2 API, enter Logtrace = Y in the *LTMCFG* file.

Note There is usually some performance impact when you use these parameters. Review the full description of a parameter in the *Replication Agent for DB2 UDB Installation Guide* before using it.

- If you need additional tracing to help debug the information passed to a Replication Agent user exit, set the value of the API_com_test configuration parameter to Y. You can also use this trace when no exit is being used.
- The LTL_test_only configuration parameter controls whether LTM for z/OS connects to Replication Server and sends transaction operations for replication. When the value of the LTL_test_only parameter is Y, LTL that would normally be sent to Replication Server is written to the *LTLOUT* file instead.

Note The Replication Agent for DB2 UDB is “not corrected to” the Replication Server when the value of the LTL_test_only parameter is Y.

- The trace=LTLebcdic configuration parameter writes EBCDIC LTL that is passed to Replication Server to *LTLOUT*. If you are replicating a table that contains ASCII data, set the trace = LTLASCII to write the ASCII characters to the *LTLOUT* data set. You must set the value of these parameters to Y to turn on this trace.
- The Use_repdef configuration parameter allows LTM for z/OS to send LTL to Replication Server that contains only the columns specified in the replication definition.

  Setting the value of the Use_repdef parameter to N may increase the amount of information provided in an LTL trace.
- The suppress_col_names configuration parameter determines whether LTM for z/OS suppresses column names from the LTL that is sent to Replication Server.

  If you are tracing LTL output, set the value of suppress_col_names to N to ensure that column names are present in the generated LTL.

Replication Agent

You can use the trace flags and configuration parameters described in this section to obtain additional information that is not normally presented by the Replication Agent (for Microsoft SQL Server, Oracle, and UDB).

Note Some performance impact usually occurs when you use these trace flags and parameters. Before using a flag or parameter, review its full description in the *Replication Agent Administration Guide*.

Trace flags

Normal trace output is sent to the Replication Agent instance log file. However, output from the LTITRACELTL trace point is sent to a separate LTL output log file (*LTITRACELTL.log*).

The following trace flags are particularly useful for troubleshooting Replication Agent problems:

- LRTRACE – traces general execution of the Log Reader component.
- LTITRACE – traces general execution of the Log Transfer Interface component.
- LTITRACELTL – enables LTL statement tracing in the *LTITRACELTL.log* file.
- RACONTRC – traces connection and query execution.

- RACONTRCSQL – traces SQL statements sent to the primary database.

Configuration parameters

The settings of the following Replication Agent configuration parameters affect the trace information:

- compress_ltl_syntax – when set to false, provides more verbose description of LTL commands.
- connect_to_rs – when set to false, allows LTL to be generated without actual connection or sending information to Replication Server.
- log_trace_verbose – when set to true, provides more verbose description of traced components.
- use_rssd – when set to false, provides a complete generation of LTL commands without modification for replication definition information.
- column_compression – when set to false, sends complete column information (all columns in after images) in the generated LTL for update operations.

For a complete description of Replication Agent trace flags and configuration settings, refer to the Replication Agent *Administration Guide*.

APPENDIX E

Reference Implementation for Oracle to Oracle Replication

This appendix describes how you can quickly set up an Oracle to Oracle reference replication environment.

Introduction

Replication Server includes a toolset for quickly setting up a reference implementation of Oracle to Oracle replication using the products available in your environment. You can implement a replication environment as a reference to demonstrate Replication Server features and functionalities.

Use the toolset to:

1 Build Replication Server and the primary and replicate databases.

2 Configure the database replication environment.

3 Perform simple transactions on the primary database and replicate the changes using database-level replication.

4 Collect statistics and monitors counters from the replication processing in step 3.

5 Clean up the reference replication environment.

The reference implementation toolset consists of scripts that are in *$SYBASE/REP-15_5/REFIMP-01_0*.

Note The reference implementation provides only a single Replication Server, primary database server, and replicate database server. You cannot configure the reference environment topology for multiple replication system components.

See Appendix D, “Implementing a Reference Replication Environment” in the *Replication Server Administration Guide Volume 2* for requirements, instructions, a sample configuration file, and the objects created by implementing the reference environment.

Platform support

You can implement a reference environment on all platforms that Replication Server supports except for Linux on IBM p-Series (Linux on Power) 64-bit. However, to set up the reference environment on any Microsoft Windows platform that Replication Server supports, you must use Cygwin to run the reference implementation scripts. See the Cygwin Web site at http://www.cygwin.com/.

Components for reference implementation

You must have supported versions of the components of a replication environment before you can implement a reference environment.

Oracle

You can build a reference implementation environment for Oracle to Oracle replication with the versions of Replication Server, Oracle, Replication Agent for Oracle, and ECDA Option for Oracle listed in Table E-1.

Table E-1: Supported product component versions for Oracle reference implementation

Replication Server	Oracle	Replication Agent for Oracle	ECDA Option for Oracle
15.5	10.2	15.2	15.0 ESD #3

For example, you can build a reference implementation environment for Oracle with Replication Server 15.5, Oracle 10.2, Replication Agent 15.2 for Oracle, and ECDA Option for Oracle 15.0 ESD #3.

Glossary

Adaptive Server — The Sybase version 11.5 and later relational database server.

applied function — A replicated function, associated with a function replication definition, that Replication Server delivers from a primary database to a subscribing replicate database. An applied function passes parameter values to a stored procedure that is executed at the replicate database. See also **replicated function delivery**, **request function**, and **function replication definition**.

asynchronous procedure delivery — A method of replicating, from a source to a destination database, a stored procedure that is associated with a table replication definition.

asynchronous command — A command that a client submits when the client is not prevented from proceeding with other operations before the completion status is received. Many Replication Server commands function as asynchronous commands within the replication system.

atomic materialization — A materialization method that copies subscription data from a primary database to a replicate database through the network in a single atomic operation, using a select operation with a holdlock. No changes to primary data are allowed until data transfer is complete. Replicate data may be applied either as a single transaction or in increments of ten rows per transaction, which ensures that the replicate database transaction log does not fill. Atomic materialization is the default method for the create subscription command. See also **nonatomic materialization**, **bulk materialization**, and **no materialization**.

autocorrection — A setting applied to replication definitions, using the set autocorrection command, to prevent failures caused by missing or duplicate rows in a copy of a replicated table. When autocorrection is enabled, Replication Server converts each update or insert operation into a delete followed by an insert. Enable autocorrection only for replication definitions whose subscriptions use non atomic materialization.

base class — A function-string class that does not inherit function strings from a parent class. See also **function-string class**.

bitmap subscription A type of subscription that replicates rows based on bitmap comparisons. Create columns using the int datatype, and identify them as the rs_address datatype when you create a replication definition. When you create a subscription, compare each rs_address column to a bitmask using a bitmap comparison operator (&) in the where clause. Rows that match the subscription's bitmap are replicated.

bulk materialization A materialization method whereby subscription data in a replicate database is initialized outside of the replication system. For example, data may be transferred from a primary database using media such as magnetic tape, CD, or optical storage disk. 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**, **nonatomic materialization**, and **no materialization**.

class See **error class** and **function-string class**.

class tree A set of function-string classes, consisting of two or more levels of derived and parent classes that derive from the same base class. See also **function-string class**.

client A program connected to a server in a client/server architecture. A may be a front-end application program executed by a user, or a utility program that executes as an extension of the system.

Client/Server Interfaces (C/SI) The Sybase interface standard for programs executing in a client/server architecture.

connection A connection from a Replication Server to a database. See also **Data Server Interface (DSI)**.

connection profiles Connection profiles allow you to configure your connection with a pre-defined set of properties.

coordinated dump A set of database dumps or transaction dumps that is synchronized across multiple sites by distributing an rs_dumpdb or rs_dumptran function through the replication system.

database A set of related data tables and other objects that is organized and presented to serve a specific purpose.

database replication definition A description of a set of database objects—tables, transactions, functions, system stored procedures, and DDL—for which a subscription can be created.

You can also create table replication definitions and function replication definitions. See also **table replication definition** and **function replication definition**.

database server A server program, such as Sybase Adaptive Server Enterprise, that provides database management services to clients.

data definition language (DDL) The set of commands in a query language, such as Transact-SQL, that describes data and their relationships in a database. DDL commands in Transact-SQL include those using the create, drop, and alter keywords.

data manipulation language (DML) The set of commands in a query language, such as Transact-SQL, that operates on data. DML commands in Transact-SQL include select, insert, update, and delete.

data server A server whose client interface conforms to the Sybase Client/Server Interfaces and provides the functionality necessary to maintain the physical representation of a replicated 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.

Data Server Interface (DSI) Replication Server threads that correspond to a connection between a Replication Server and a database. DSI threads submit transactions from the DSI outbound queue to a replicate data server. They consist of a scheduler thread and one or more executor threads. The scheduler thread groups the transactions by commit order and dispatches them to the executor threads. The executor threads map functions to function strings and execute the transactions in the replicate database. DSI threads use an Open Client connection to a database. See also **outbound queue** and **connection**.

data source A specific combination of a database management system (DBMS) product such as a relational or non-relational data server, a database residing in that DBMS, and the communication method used to access that DBMS from other parts of a replication system. See also **database** and **data server**.

declared datatype The datatype of the value delivered to the Replication Server from the Replication Agent:

- If the Replication Agent delivers a base Replication Server datatype (such as datetime) to the Replication Server, the declared datatype is the base datatype.
- Otherwise, the declared datatype must be the user-defined datatype (UDD) for the original datatype at the primary database.

default function string — The function string that is provided by default for the system-provided classes rs_sqlserver_function_class and rs_default_function_class and classes that inherit function strings from these classes, either directly or indirectly. See also **function string**.

dematerialization — The optional process when a subscription is dropped, whereby specific rows that are not used by other subscriptions are removed from the replicate database.

derived class — A function-string class that inherits function strings from a parent class. See also **function-string class** and **parent class**.

distributed database system — A database system where data is stored in multiple databases on a network. The databases may be managed by data servers of the same type (for example, Adaptive Server) or by heterogeneous data servers.

Distributor — A Replication Server thread (DIST) that helps to determine the destination of each transaction in the inbound queue.

dump marker — A message written by Adaptive Server in a database transaction log when a dump is performed. In a warm standby application, when you are initializing the standby database with data from the active database, you can specify Replication Server to use the dump marker to determine where in the transaction stream to begin applying transactions in the standby database.

Enterprise Connect Data Access (ECDA) — An integrated set of software applications and connectivity tools that allow access to data within a heterogeneous database environment, such as a variety of LAN-based, non-ASE data sources, and mainframe data sources.

ExpressConnect for Oracle — A set of libraries that can be used to provides direct communication between Replication Server and an Oracle database.

error action — A Replication Server response to a data server error. Possible Replication Server error actions are ignore, warn, retry_log, log, retry_stop, and stop_replication. Error actions are assigned to specific data server errors.

error class — A name for a collection of data server error actions that are used with a specified database.

function — A Replication Server object that represents a data server operation, such as insert, delete, select, or begin transaction. Replication Server distributes such operations to other Replication Servers as functions. Each function consists of a function name and a set of data parameters. To execute the function in a destination database, Replication Server uses function strings to convert a function to a command or set of commands for a type of database. See also **user-defined** and **replicated function delivery**.

function replication definition A description of a replicated function used in replicated function delivery. The function replication definition, maintained by Replication Server, includes information about the parameters to be replicated and the location of the primary version of the affected data. See also **replicated function delivery**.

function scope The range of a function's effect. Functions have replication definition scope or function-string class scope. A function with replication definition scope is defined for a specific replication definition and cannot be applied to other replication definitions. A function with function-string class scope is defined once for a function-string class and is available only within that class.

function string A string that Replication Server uses to map a database command 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 functions, the string also contains an output template, used to format the database command for the destination data server.

function-string class A named collection of function strings used with a specified database connection. Function-string classes include those provided with Replication Server and those you have created. Function-string classes can share function string definitions through function-string inheritance. See also **base class**, **class tree**, **derived class**, **function-string inheritance**, and **parent class**.

function-string inheritance The ability to share function string definitions between classes, whereby a derived class inherits function strings from a parent class. See also **derived class**, **function-string class**, and **parent class**.

function-string variable An identifier used in a function string to represent a value that is to be substituted at runtime. Variables in function strings are enclosed in question marks (?). They represent column values, function parameters, system-defined variables, or user-defined variables.

function subscription A subscription to a function replication definition (used in applied function delivery).

gateway Connectivity software that allows two or more computer systems with different network architectures to communicate.

heterogeneous data servers Data servers that are supplied by more than one vendor used together in a distributed database system.

interface file — A file containing entries that define network access information for server programs in a Sybase client/server architecture. Server programs may include Adaptive Servers, SQL Servers, gateways, Replication Servers, and Replication Agents such as LTM for SQL Server. The *interfaces* file entries enable clients and servers to connect to each other in a network.

latency — The measure of the time it takes to distribute to a replicate database a data modification operation first applied in a primary database. The time includes Replication Agent processing, Replication Server processing, and network overhead.

locator value — The value stored in the rs_locater table of the Replication Servers RSSD that identifies the latest log transaction record received and acknowledged by the Replication Server from each previous site during replication.

login name — The name that a user or a system component such as Replication Server uses to log in to a data server, Replication Server, or Replication Agent.

Log Transfer Language (LTL) — A subset of the Replication Command Language (RCL). A Replication Agent such as RepAgent or LTM for SQL Server uses LTL commands to submit to Replication Server the information it retrieves from primary database transaction logs.

Log Transfer Manager (LTM) — The Replication Agent program for Sybase SQL Server. See also **Replication Agent** and **Replication Agent thread**.

maintenance user — A data server login name that Replication Server uses to maintain replicate data. In most applications, maintenance user transactions are not replicated.

materialization — Copying data specified by a subscription from a primary database to a replicate database, thereby initializing the replicate table. Replicate data can be transferred over a network, or, for subscriptions involving large amounts of data, loaded initially from media. See also **atomic materialization**, **bulk materialization**, **no materialization**, and **nonatomic materialization**.

materialization queue — A stable queue used to spool messages related to a subscription being materialized or dematerialized.

missing row — A row missing from a replicated copy of a table but present in the primary table.

multi-site availability (MSA) — Methodology for replicating database objects—tables, functions, transactions, system stored procedures, and DDL from the primary to the replicate database. See also **database replication definition**.

namespace — The scope within which an object name must be unique.

nonatomic materialization
A materialization method that copies subscription data from a primary to a replicate database through the network in a single operation, without a holdlock. Changes to the primary table are allowed during data transfer, which may cause temporary inconsistencies between replicate and primary databases. Data is applied in increments of ten rows per transaction, which ensures that the replicate database transaction log does not fill. Nonatomic materialization is an optional method for the create subscription command. See also **autocorrection**, **atomic materialization**, **no materialization**, and **bulk materialization**.

network-based security
Secure transmission of data across a network. Replication Server supports third-party security mechanisms that provide user authentication, unified login, and secure message transmission between Replication Servers.

no materialization
A materialization method that lets you create a subscription when the subscription data already exists at the replicate site. Use the create subscription command with the without materialization clause. Use the no-materialization method to create subscriptions to table replication definitions and function replication definitions. See also **atomic materialization** and **bulk materialization**.

online transaction processing (OLTP) application
A database client application characterized by frequent transactions involving data modification (inserts, deletes, and updates).

origin queue ID (QID)
Formed by the Replication Agent, the queue ID (qid) uniquely identifies each log record passed to the Replication Server. It includes the date and time from the primary data server, and the database generation number.

orphaned row
A row in a replicated copy of a table that does not match an active subscription.

outbound queue
A stable queue used to spool messages. The DSI outbound queue spools messages to a replicate database. The RSI outbound queue spools messages to a replicate Replication Server.

parallel DSI
A method of configuring a database connection so that transactions are applied to a replicate data server using multiple DSI threads operating in parallel, rather than a single DSI thread. See also **connection** and **Data Server Interface (DSI)**.

parameter
An identifier representing a value that is provided when a procedure executes. Parameter names are prefixed with an @ character in function strings. When a procedure is called from a function string, Replication Server passes the parameter values, unaltered, to the data server. See also **searchable parameter**.

parent class
A function-string class from which a derived class inherits function strings. See also **function-string class** and **derived class**.

primary data The definitive version of a set of data in a replication system. The primary data is maintained on a data server that is known to all of the Replication Servers with subscriptions for the data.

primary database Any database that contains data that is replicated to another database through the replication system.

primary key A set of table columns that uniquely identifies each row.

principal user The user who starts an application. When using network-based security, Replication Server logs in to remote servers as the principal user.

profiles Profiles allow you to configure your connection with a pre-defined set of properties.

publication A group of articles from the same primary database. A publication lets you collect replication definitions for related tables and stored procedures and then subscribe to them as a group. Collect replication definitions as articles in a publication at the source Replication Server and subscribe to them with a publication subscription at the destination Replication Server.

publication subscription A subscription to a publication. See also **publication**.

published datatype The datatype of the column after the column-level translation (and before a class-level translation, if any) at the replicate data server. The published datatype must be either a Replication Server base datatype or a UDD for the datatype in the target data server. If the published datatype is omitted from the replication definition, it defaults to the declared datatype

query In a database management system, a request to retrieve data that meets a given set of criteria. The SQL database language includes the select command for queries.

quiescent A replication system in which all data-changing operations (or transactions) have been propagated to their destinations. Some Replication Server commands or procedures require that you first quiesce the replication system.

remote procedure call (RPC) A request to execute a procedure that resides in a remote server. The server that executes the procedure be an Adaptive Server, a Replication Server, or a server created using Open Server. The request can originate from any of these servers or from a client application. The RPC request format is a part of the Sybase Client/Server Interfaces.

Replication Agent thread — The Replication Agent for Adaptive Server Enterprise. The Replication Agent thread is an Adaptive Server thread. It sends transaction log information from the primary database to the primary Replication Server.

Replication Agent User thread — The thread on a Replication Server database connection that a Replication Agent connects with, on behalf of a primary database. See also **Data Server Interface (DSI)**.

replicate database — Any database that contains data that is replicated from another database through the replication system.

replicated function delivery — A method of replicating, from a source to a destination database, a stored procedure that is associated with a function replication definition. See also **applied function**, **request function**, and **function replication definition**.

replicated stored procedure — An Adaptive Server stored procedure that is marked as replicated using the sp_setrepproc or the sp_setreplicate system procedure. Replicated stored procedures can be associated with function replication definitions or table replication definitions. See also **replicated function delivery** and **asynchronous procedure delivery**.

replicated table — A table that is maintained by Replication Server, in part or in whole, in databases at multiple locations. There is one primary version of the table, which is marked as replicated using the sp_setreptable or the sp_setreplicate system procedure; all other versions are replicated copies.

Replication Agent — A program or module that sends transaction log information from a primary data server to a primary Replication Server. Sybase provides separate Replication Agent software products to support non-ASE data servers in a replication system.

Replication Command Language (RCL) — The commands used to manage information in Replication Server.

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.

You can also create function replication definitions; sometimes the term “table replication definition” is used to distinguish between table and function replication definitions. See also **function replication definition**.

Replication Server — The Sybase server program that maintains replicated data, typically on a LAN, and processes data transactions received from other Replication Servers on the same LAN or on a WAN.

Replication Server Interface (RSI) A thread that logs in to a destination Replication Server and transfers commands from the RSI outbound stable queue to the destination Replication Server. There is one RSI thread for each destination Replication Server that is a recipient of commands from a primary or intermediate Replication Server. See also **outbound queue**.

Replication system administrator The system administrator who manages routine operations in the Replication Server.

Replication Server System Database (RSSD) The Adaptive Server database containing a Replication Server system tables. You can choose whether to store Replication Server system tables on the RSSD or the Adaptive Server® Anywhere (ASA) ERSSD.

Replication Server system Adaptive Server The Adaptive Server with the database containing a Replication Servers system tables (the RSSD).

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 on Replication Server and includes other components, such as Replication Agents and data servers.

replication system domain All replication system components that use the same ID Server.

request function A replicated function, associated with a function replication definition, that Replication Server delivers from a replicate database to a primary database. A request function passes parameter values to a stored procedure that is executed at the primary database. See also **replicated function delivery**, **request function**, and **function replication definition**.

row migration The process whereby column value changes in rows in a primary version of a table cause corresponding rows in a replicate version of the table to be inserted or deleted, based on comparison with values in a subscription's where clause.

SQL Server The Sybase relational database pre-11.5 server.

schema The structure of the database. DDL commands and system procedures change system tables stored in the database. Supported DDL commands and system procedures can be replicated to standby databases when you use Replication Server version 11.5 or later and Adaptive Server version 11.5 or later.

searchable column A column in a replicated table that can be specified in the where clause of a subscription or article to restrict the rows replicated at a site.

searchable parameter — A parameter in a replicated stored procedure that can be specified in the where clause of a subscription to help determine whether or not the stored procedure should be replicated. See also **parameter**.

secondary truncation point — See **truncation point**.

site — An installation consisting of, at minimum, a Replication Server, data server, and database, and possibly a Replication Agent, usually at a discrete geographic location. The components at each site are connected over a WAN to those at other sites in a replication system.

site version — The version number for an individual Replication Server. Once the site version has been set to a particular level, the Replication Server enables features specific to that level, and downgrades are not allowed. See also **software version**, and **system version**.

software version — The version number of the software release for an individual Replication Server. See also **site version** and **system version**.

Stable Queue Manager (SQM) — A thread that manages the stable queues. There is one Stable Queue Manager (SQM) thread for each stable queue accessed by the Replication Server, whether inbound or outbound.

Stable Queue Transaction (SQT) interface — A thread that reassembles transaction commands in commit order. A Stable Queue Transaction (SQT) interface thread reads from inbound stable queues, puts transactions in commit order, then sends them to the Distributor (DIST) thread or a DSI thread, depending on which thread required the SQT ordering of the transaction.

stable queues — Store-and-forward queues where Replication Server stores messages destined for a route or database connection. Messages written into a stable queue remain there until they can be delivered to the destination Replication Server or database. Replication Server builds stable queues using its disk partitions. See also **outbound queue**, and **materialization queue**.

stored procedure — A collection of SQL statements and optional control-of-flow statements stored under a name in an Adaptive Server database. Stored procedures supplied with Adaptive Server are called system procedures. Some stored procedures for querying the RSSD are included with the Replication Server software.

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. You can also subscribe to a function replication definition, for replicating stored procedures.

subscription dematerialization — See **dematerialization**.

subscription materialization — See **materialization**.

subscription migration — See **row migration**.

Sybase Central — A graphical tool that provides a common interface for managing products. Replication Server uses Replication Server Manager as a Sybase Central plug-in.

synchronous command — A command that a client considers complete only after the completion status is received.

system function — A function that is predefined and part of the Replication Server product. Different system functions coordinate replication activities, such as rs_begin, or perform data manipulation operations, such as rs_insert, rs_delete, and rs_update.

system-provided classes — The error class includes non-ASE error classes and the function-string classes that Replication Server provides. Function strings are generated automatically for the system-provided function-string classes and for any derived classes that inherit from these classes, directly or indirectly. See also **error class** and **function-string class**.

system version — The version number for a replication system that represents the version for which new features are enabled, for Replication Servers of 11.0.2 or earlier, and below which no Replication Server can be downgraded or installed. For a Replication Server version 11.5, use of certain new features requires a site version of 1150 and a system version of at least 1102. See also **site version**, and **software version**.

table replication definition — See **replication definition**.

table subscription — A subscription to a table replication definition.

thread — A process running within Replication Server. Built upon Sybase Open Server, Replication Server has a multi-threaded architecture. Each thread performs a certain function such as managing a user session, receiving messages from a Replication Agent or another Replication Server, or applying messages to a database. See also **Data Server Interface (DSI)**, **Distributor**, and **Replication Server Interface (RSI)**.

transaction A mechanism for grouping statements so that they are treated as a unit: either all statements in the group are executed or no statements in the group are executed.

Transact-SQL The relational database language used with Adaptive Server. Transact-SQL is based on standard SQL (Structured Query Language), with Sybase extensions.

truncation point An Adaptive Server database that holds primary data has an active truncation point, marking the transaction log location where Adaptive Server has completed processing. This is the primary truncation point.

The Replication Agent for an Adaptive Server database maintains a secondary truncation point, marking the transaction log location separating the portion of the log successfully submitted to the Replication Server from the portion not yet submitted. The secondary truncation point ensures that each operation enters the replication system before its portion of the log is truncated.

user-defined A function that allows you to create custom applications that use Replication Server to distribute replicated functions or asynchronous stored procedures between sites in a replication system. In replicated function delivery, a user-defined function is automatically created by Replication Server when you create a function replication definition.

variable See **function-string variable**.

version See **site version**, **software version**, and **system version**.

warm standby application A **warm standby application** is a pair of databases, one of which is a backup copy of the other. Client applications update the **active database**; Replication Server maintains the **standby database** as a copy of the active database.

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

Z