SYBASE®
An SAP® Company

Heterogeneous Replication Guide

Replication Server® 15.7.1 SP100

DOCUMENT ID: DC36924-01-1571100-01
LAST REVISED: May 2013

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

Contents

Conventions

These style and syntax conventions are used in Sybase® documentation.

Style conventions

Key	Definition
`monospaced(fixed-width)`	• SQL and program code • Commands to be entered exactly as shown • File names • Directory names
`italic monospaced`	In SQL or program code snippets, placeholders for user-specified values (see example below).
italic	• File and variable names • Cross-references to other topics or documents • In text, placeholders for user-specified values (see example below) • Glossary terms in text
bold san serif	• Command, function, stored procedure, utility, class, and method names • Glossary entries (in the Glossary) • Menu option paths • In numbered task or procedure steps, user-interface (UI) elements that you click, such as buttons, check boxes, icons, and so on

If necessary, an explanation for a placeholder (system- or setup-specific values) follows in text. For example:

Run:

```
installation directory\start.bat
```

where *installation directory* is where the application is installed.

Syntax conventions

Key	Definition
{ }	Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces when you enter the command.
[ ]	Brackets mean that choosing one or more of the enclosed options is optional. Do not type the brackets when you enter the command.
( )	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.
...	An ellipsis (three dots) means you may repeat the last unit as many times as you need. Do not include ellipses in the command.

Case-sensitivity

- All command syntax and command examples are shown in lowercase. However, replication command names are not case-sensitive. For example, **RA_CONFIG**, **Ra_Config**, and **ra_config** are equivalent.
- Names of configuration parameters are case-sensitive. For example, **Scan_Sleep_Max** is not the same as **scan_sleep_max**, and the former would be interpreted as an invalid parameter name.
- Database object names are not case-sensitive in replication commands. However, to use a mixed-case object name in a replication command (to match a mixed-case object name in the primary database), delimit the object name with quote characters. For example: **pdb_get_tables** "*TableName*"
- Identifiers and character data may be case-sensitive, depending on the sort order that is in effect.
  - If you are using a case-sensitive sort order, such as “binary,” you must enter identifiers and character data with the correct combination of uppercase and lowercase letters.
  - If you are using a sort order that is not case-sensitive, such as “nocase,” you can enter identifiers and character data with any combination of uppercase or lowercase letters.

Terminology

Replication Agent™ is a generic term used to describe the Replication Agents for Adaptive Server® Enterprise, Oracle, IBM DB2 UDB, and Microsoft SQL Server. The specific names are:

- RepAgent – Replication Agent thread for Adaptive Server Enterprise
- Replication Agent for Oracle

- Replication Agent for Microsoft SQL Server
- Replication Agent for UDB – for IBM DB2 on Linux, Unix, and Windows
- Replication Agent for DB2 for z/OS

Replication System Overview

Sybase supports a basic replication system from an Adaptive Server Enterprise (ASE) server to another ASE server, and a heterogeneous replication system, where one or more servers is not an ASE.

Basic Replication System

A basic Sybase replication system consists of a primary Adaptive Server Enterprise (ASE) database, a Replication Server®, and a replicate ASE database.

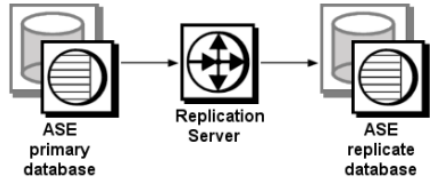
ASE includes all the features necessary to support a Sybase replication system, with no additional components other than the Replication Server.

The three components include:

- *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.

The Basic Sybase replication system diagram illustrates a basic Sybase replication system, showing the flow of data between two Adaptive Servers and a Replication Server.

Figure 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 *Replication Server Design Guide* and *Introduction to Replication Server* in the *Replication Server Administration Guide: Volume 1*.

Heterogeneous Replication System

A heterogeneous Sybase replication system consists of data-changing operations between two databases of the same or different vendors (except ASE to ASE).

For information on ASE to ASE replication, see the *Replication Server Administration Guide Volume 1 and Volume 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).

ASE 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.

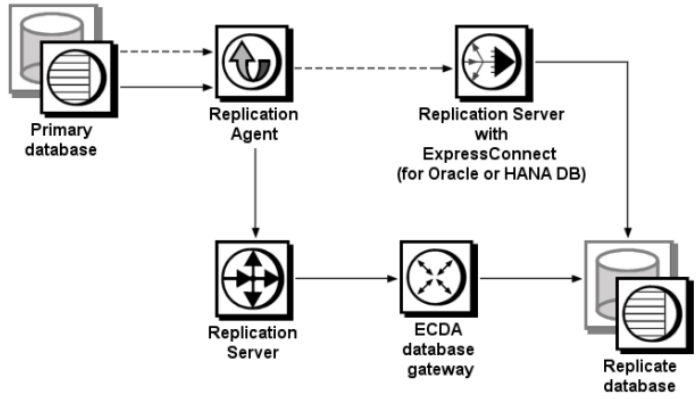
Additional components are required to implement a Sybase replication system with non-ASE data servers:

- A Replication Agent
- Enterprise Connect™ Data Access for ODBC (for Microsoft or DB2 UDB) or ExpressConnect (for Oracle or for HANA DB), which are required if the replication data server connectivity requirements are not compatible with Replication Server

Sybase Replication System with Non-ASE Data Servers diagram 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 or for HANA DB).

Figure 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 ECDA database gateway, and then from the database gateway to the replicate database.

ECDA database gateways support IBM DB2 UDB and Microsoft SQL Server by providing connectivity between Sybase Open Client and Open Server and ODBC to 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 or for HANA DB), 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.

Sybase Replication System Components

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

The replication system components include:

- Primary data server
- Replication Agent
- Replication Server
- A connector to the replicate database, such as:
  - Enterprise Connect Data Access

  - ExpressConnect for Oracle
  - ExpressConnect for HANA DB
- Replicate data server

See also

- *Sybase Replication Products* on page 25

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.

See also

- *Primary Database* on page 14

Supported Primary Database Servers

Sybase replication technology actively supports transaction replication from different relational database servers aside from Adaptive Server Enterprise.

The supported relational database servers include:

- 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.

Replication Agent

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.

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

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

For non-ASE data servers, Sybase provides these 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.

Replication Server

The 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 provides guaranteed delivery of transactions to each replicate site by:

- Receiving transactions from primary databases through a Replication Agent and distributing them to replicate database sites that have subscriptions for the data
- Receiving transactions from other Replication Servers and applying them to local replicate databases or forwarding them to other replication servers that have subscriptions for the data
- Receiving requests for data updates from a replicate database and applies them to a primary database

The information needed to accomplish these tasks is stored in Replication Server system tables that are stored in the Replication Server System Database.

See *Replication Server Administration Guide Volume 2 > Performance Tuning > Replication Server Internal Processing* for more information about the internal elements of the Replication Server.

ID Server

The ID Server is a Replication Server that registers all Replication Servers and databases in the replication system.

In addition to the usual Replication Server tasks, the Replication Server acting as the ID Server assigns a unique ID number to every Replication Server and database in the replication system. The ID Server also maintains version information for the replication system. Otherwise, the ID Server is like any other Replication Server.

To allow a new Replication Server, or the Replication Server that manages the new database, to log in and retrieve an ID number, the ID Server must be running each time a:

- Replication Server is installed
- Route is created
- Database connection is created or dropped

Because of these requirements, the ID Server is the first Replication Server that you install and start when you install a replication system. If you have only one Replication Server, or if you

are installing Replication Server for the first time, then that Replication Server is also the ID Server. If you are adding a Replication Server to an existing replication system, you must know the name of the Replication Server in the system that is the ID Server.

The ID Server must have a login name for Replication Servers to use when they connect to the ID Server. The login name is recorded in the configuration files of all Replication Servers in the replication system by the **rs_init** configuration program when you are setting up and managing the replication system.

Warning! The ID Server is critical to your replication environment, and is difficult to move once it has been installed. Once you have selected a name for the ID Server, you cannot change to a different Replication Server. Sybase does not support any procedures that change the name of the ID Server in the configuration files.

Replication System Domain

Replication system domain refers to all replication system components that use the same ID Server.

Some organizations have multiple independent replication systems. Since the ID Server determines member Replication Servers and databases in a replication system, one replication system in an organization with multiple replication systems is also called an ID Server domain.

No special steps are required to set up multiple ID Server domains. Every Replication Server or database belongs to one replication system and has a unique ID number in that ID Server domain.

You can set up multiple replication system domains, with the following restrictions:

- Replication Servers in different domains cannot exchange data. Each domain must be treated as a separate replication system with no cross-communication between them. You cannot create a route between Replication Servers in different domains.
- A database can be managed by only one Replication Server in one domain. Any given database is in one, and only one, ID Server's domain. This means that you cannot create multiple connections to the same database from different domains.

Replication Server System Database (RSSD)

The Replication Server System Database (RSSD) is a database that contains the Replication Server system tables.

Each Replication Server requires an RSSD or an Embedded Replication Server System Database (ERSSD) to hold the system tables for one Replication Server. The RSSD is managed by the Adaptive Server. The ERSSD is managed by SQL Anywhere®.

System Tables

Replication Server system tables hold information that Replication Server requires to send and receive replicated data.

System tables hold information such a:

- Descriptions of replicated data and related information
- Descriptions of replication objects, such as replication definitions and subscriptions
- Security records for Replication Server users
- Routing information for other Replication Server sites
- Access methods for the local databases
- Other administrative information

The Replication Server system tables are loaded into the RSSD during Replication Server installation.

See *Replication Server Reference Manual > Replication Server System Tables* for a comprehensive list of system tables.

System table contents are modified during Replication Server activities, such as the execution of RCL commands or Sybase Central™ procedures. Only the replication system administrator, or members of the **rs_systabgroup** group, can alter the system tables.

To query the system tables and find status information:

- Use Sybase Central to view replication system details and properties.
- Use Replication Server system information or system administration commands. See *Replication Server Reference Manual > Introduction to the Replication Command Language > System Information Commands* and *Replication Server Reference Manual > Introduction to the Replication Command Language > System Administration Commands*.
- Use Adaptive Server stored procedures to display information about the replication system. See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures*.

Warning! RSSD tables are for internal use by Replication Server only. You should never modify RSSD tables directly unless directed by Sybase Technical Support.

RSSD and Replication Agent Specifications

A Replication Agent is needed for the RSSD if the Replication Server is the source for any route.

If Replication Server is the source for any route, Replication Server distributes some of the information in its RSSD to other Replication Servers.

The RSSD is dedicated to the Replication Server that it supports; do not use it to store user data. However, a single data server may contain the RSSD and user databases. The database

device space for the RSSD must be at least 20MB (10MB for data and 10MB for the log). It is best to put the database and the database log on separate devices.

Database Gateway

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 the retrieval of metadata from non-ASE replicate data servers.

See also

- *Enterprise Connect Data Access* on page 40

ExpressConnect for Oracle

ExpressConnect for Oracle provides direct communication between Replication Server and a replicate Oracle data server.

ExpressConnect for Oracle which is available with Replication Server Options 15.5 and later, 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 also

- *ExpressConnect for Oracle* on page 44

ExpressConnect for HANA DB

ExpressConnect for HANA DB is an embedded library loaded by Replication Server for replication to HANA DB.

ExpressConnect for HANA DB is installed with Replication Server 15.7.1 SP100 to provide direct communication between Replication Server and a replicate HANA DB data server.

ExpressConnect for HANA DB is installed with Replication Server. There is no separate installer for ExpressConnect for HANA DB.

See the *Replication Server Heterogeneous Replication Guide > HANA DB as Replicate Data Server > ExpressConnect for HANA DB and Replicate Database Connectivity for HANA DB*.

See also

- *ExpressConnect for HANA DB* on page 44

Replicate Data Server

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).

See also

- *Replicate Database* on page 14

Supported Replicate Database Servers

Sybase replication technology supports transaction replication in different relational database servers.

The relational database servers include :

- HANA DB
- 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 Microsoft SQL Server and DB2 UDB data servers, see 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 Configuration Guide*. For information on ExpressConnect for HANA DB, see "HANA DB as Replicate Data Server" in this guide.

Non-ASE Replication

When replicating with non-ASE servers, you must consider issues that are specific to the data server's role in the replication system regardless of the type or brand of a data server.

The biggest challenge in implementing a successful heterogeneous replication system is accommodating the unique characteristics of data servers that are supplied by different vendors. 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

There are primary database issues that must be addressed in a successful heterogeneous replication system.

When using a non-ASE primary database, consider:

- 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 topic for your database.

Replicate Database

There are replicate database issues that must be addressed in a successful heterogeneous replication system.

When using a non-ASE replication database, consider:

- Whether or not an ExpressConnect is available for your replicate database. To simplify your replication system and improve performance, use ExpressConnect when it is available for your replicate database.

- 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 and ExpressConnect (for Oracle or HANA DB) use either the native communication protocol of a data server or the standard ODBC protocol to communicate with a replicate database. ECDA gateways and ExpressConnect (for Oracle or HANA DB) require a connectivity library that you must provide. Replication Server generally requires a database gateway or ExpressConnect (for Oracle or HANA DB) 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 topic for your database.

Character Sets

Setting character sets avoid problem that can produce data inconsistencies between the primary database and the replicate database.

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.

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

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, ECDA database gateway, and ExpressConnect (for Oracle or HANA DB). Refer to the documentation for the specific Replication Agent, ECDA, and ExpressConnect 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

The LOB datatypes impact the transaction logging function at the primary database.

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

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 to Microsoft and the Mainframe Connect™ DirectConnect for z/OS Option gateways provide this feature. The ECDA Option for ODBC to DB2 UDB does not provide this feature. ExpressConnect (for Oracle or HANA DB) does not require support for text pointer processing to support LOB replication. ExpressConnect (for Oracle or HANA DB) supports LOB replication but not text pointer processing, which is not required.

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.

The ECDA Option for ODBC provides support for LOB replication into Microsoft SQL Server databases.

Setup for Replicate Databases

Replication Server provides a utility named **rs_init**, which sets up Adaptive Server databases.

rs_init 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.

rs_set_ciphertext Function String

rs_set_ciphertext controls replication of encrypted columns to an Adaptive Server table.

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 **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 **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 **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

The **rs_subcmp** is an 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**, provided 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.

You can also use Replication Server Data Assurance Option to compare row data and schema between two or more databases, and report discrepancies.

See also

- *Heterogeneous Database Reconciliation* on page 263

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, HANA DB, Microsoft SQL Server, DB2 UDB z/OS, and DB2 UDB on UNIX, Windows, and Linux. It is not available for 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 Sybase IQ, Oracle, and HANA DB when using ExpressConnect (for Oracle or HANA DB).

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, HANA DB, 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

Replication system issues may arise due to different configurations with heterogeneous or non-ASE data servers.

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 *Replication Server Administration Guide Volume 1 > Manage Replicated Tables > Translating Datatypes Using HDS*.

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 replicates one-way from an Adaptive Server primary database to a non-ASE replicate.

The only unique requirements are a component 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 *Replication Server Administration Guide Volume 1 > Manage Replicated Tables > Translating Datatypes Using HDS*.

Replication System Components

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

- Adaptive Server primary database
- Replication Server
- ExpressConnect (for Oracle or HANA DB) to replicate to Oracle or HANA DB; or a relevant ECDA database gateway designed for the replicate data server, such as ECDA Option for ODBC for Microsoft SQL Server
- Non-ASE replicate data server. For example, Microsoft SQL Server

Replication System Issues

In an Adaptive Server primary to non-ASE replicate configuration, consider:

- 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.
- Create the Replication Server replicate database connection using the correct profile for the primary and 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

The Non-ASE primary to Non-ASE replicate 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 replication 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.
- ExpressConnect (for Oracle or HANA DB) to replicate to Oracle or HANA DB, or a relevant ECDA database gateway designed for the replicate data server, such as the 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 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).
- To use direct-load materialization, you must create the connection to the primary database using the appropriate connection profile: rs_rs_to_oracle_ra, rs_rs_to_mssql_ra, rs_rs_to_udb_ra, or rs_rs_to_hanadb_ra.
- The Replication Server replicate database connection must be created using the correct profile for the primary and 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

Replication occurs both to and from each database in a bidirectional non-ASE to non-ASE replication scenario.

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.
- ExpressConnect (for Oracle or HANA DB) to replicate to Oracle or HANA DB, or a relevant ECDA database gateway designed for the replicate data server, such as the ECDA Option for ODBC (for DB2 UDB).
- Replication Server.
- ExpressConnect (for Oracle or HANA DB) to replicate to Oracle or HANA DB, or a relevant ECDA database gateway designed for the replicate data server, such as the ECDAOption 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:

- To use direct-load materialization, you must have two connections for each database: one for incoming data and one for outgoing data. You must create the connection for incoming data—which is used by Replication Agent—using the appropriate connection profile (rs_rs_to_*<replicate>*_ra, where *<replicate>* is oracle, mssql, udb, or hanadb). Use a connection profile supporting normal replication to create the connection for outgoing data.
- 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.

Sybase Replication Products

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

Sybase Replication products include:

- 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 Enterprise Connect Data Access (ECDA), or ExpressConnect (for Oracle or HANA DB).
  - 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 or HANA DB) supports Replication Server by providing direct communication between Replication Server and an Oracle or HANA DB database, without a need for a separate gateway server. ExpressConnect for Oracle is only available with Replication Server Options 15.5 or later. ExpressConnect for HANA DB is only available with Replication Server 15.7.1 SP100 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 distributes data over a network by managing replicated transactions while retaining transaction integrity across the network.

It also provides application developers and system administrators with a flexible publish-and-subscribe model for marking data and stored procedures to be replicated.

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

Data are published at the primary sites to which Replication Servers at other (replicate sites) subscribe.

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. To publish and subscribe data, 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

Performance can be improved over normal data replication by encapsulating many changes in a single replicated function.

Because they are not associated with table replication definitions, replicated functions can execute stored procedures that may or may not modify data directly.

With some data servers, Replication Server allows you to replicate stored procedure invocations asynchronously between databases.

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 transaction-processing services. To guarantee the integrity of distributed data, data servers must comply with transaction-processing conventions, such 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 replicate database directly, when using ExpressConnect (for Oracle or HANA DB).

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. The gateway software for replicating to DB2 UDB and Microsoft SQL Server is in the form of a Sybase ECDA database gateway. ECDA

gateways bridge from Open Server/TDS to an ODBC driver for the data server. Replication Server configurations vary, depending on the gateway used.

Replication to Oracle or HANA DB using ExpressConnect does not require an additional gateway; ExpressConnect uses native Oracle or HANA DB connectivity, allowing Replication Server using ExpressConnect to connect directly to the replicate database.

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.

You can define user IDs 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.

These are the user IDs that 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 (with 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 *Replication Agent Administration Guide*.

rs_ticket

Some Replication Agents can start **rs_ticket** transactions.

The 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 or HANA DB, 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 or HANA DB 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

The maintenance user inserts, deletes, and updates rows in replicated tables, and executes replicated stored procedures. The database owner (or system administrator) must grant the permission required for the maintenance user to perform these tasks.

To update replicated data, Replication Server logs in to the replicate data server as the maintenance user. 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, Datatype Definitions, and Restricted Datatypes

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

For more information about datatype definitions (user-defined datatypes), see *Datatype Translation and Mapping* on page 229.

Restricted Datatype

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

Error Classes for Non-ASE Data Servers

Replication Server provides error classes 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.

Function-String Classes for Non-ASE Data Servers

Replication Server provides function-string classes and associated function strings for all supported non-ASE replicate data servers.

After you create a function-string class derived from a non-ASE database parent class, manually update the `rs_translation` table in the Replication Server System Database (RSSD) to ensure that the derived class inherits the class-level translations from the parent class.

See *Function-String Inheritance* in the *Replication Server Administration Guide Volume 2*.

Updating the rs_translation Table for Derived Function-String Classes

This example shows you how to update the `rs_translation` table in the Replication Server System Database (RSSD) to ensure that the derived class you created inherits the class-level translations from the parent class.

1. Create the derived function-string class in Replication Server.
   For example, to create the **private_class_for_oracle** derived function-string class for Oracle, enter:

   ```
   create function string class private_class_for_oracle
   set parent to rs_oracle_function_class
   go
   ```

   Ensure that you specify the appropriate non-ASE database base class as your parent class when you create the derived class. See *Function-String Inheritance* in the *Replication Server Administration Guide Volume 2*.
2. Use an SQL **select** statement to obtain the class ID of **private_class_for_oracle** from the `rs_classes` Replication Server system table:

```
select classid from rs_classes where
classname='private_class_for_oracle'
go
```

You see:

```
classid
------------------
0x0100006801000065

(1 row affected)
```

3. Obtain the class-level translations defined for **rs_oracle_function_class**:

```
select * from rs_translation where classid = 0x0000000001000007
go
```

You see (split into two sets of four columns for clarity):

```
prsid classid            type source_dtid
----- ------------------ ---- ------------------
    0 0x0000000001000007 D    0x0000000000000001
    0 0x0000000001000007 D    0x000000000000000c
    0 0x0000000001000007 D    0x000000000000000d
    0 0x0000000001000007 D    0x000000000000000e
    0 0x0000000001000007 D    0x000000000000000f
    0 0x0000000001000007 D    0x0000000000000013
    0 0x0000000001000007 D    0x000000000000001b
    0 0x0000000001000007 D    0x000000000000001c

target_dtid        target_length target_status rowtype
------------------ ------------- ------------- -------
0x0000000000010202 0             0             0
0x0000000000010200 19            0             0
0x0000000000010200 19            0             0
0x0000000000010205 136           0             0
0x0000000000010205 136           0             0
0x0000000000010202 0             0             0
0x0000000000010201 9             0             0
0x0000000000010213 8             0             0

(8 row affected)
```

4. For each row returned in step 3, build SQL **insert** statements to insert the class-level translation values you obtained in step 3 into the **private_class_for_oracle** derived function-string class.
   Use the template:

```
insert into rs_translation
(prsid,
classid,
type,
source_dtid,
target_dtid,
target_length,
target_status,
```

```
rowtype)
values (0,
classid from step 2,
'D',
source_dtid from output of 3,
target_dtid from output of step 3,
target_length from output of step 3,
target_status from output of 3,
0)
```

See *rs_translation* in the *Replication Server Reference Manual*.

For example, the **insert** statement for the first row for the **private_class_for_oracle** derived function-string class should be:

```
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000000c, 0x0000000000010200, 19, 0, 0)
```

5. Execute the **insert** statements you built in step 4:

```
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000000c, 0x0000000000010200, 19, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000000d, 0x0000000000010200, 19, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x0000000000000001, 0x0000000000010202, 0, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x0000000000000013, 0x0000000000010202, 0, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000000E, 0x0000000000010205, 136, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000000f, 0x0000000000010205, 136, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000001b, 0x0000000000010201, 9, 0, 0)
insert rs_translation values (0, 0x0100006801000065, 'D',
0x000000000000001c, 0x0000000000010213, 8, 0, 0)
```

6. Restart Replication Server if the `rs_translation` rows for your classid were already in the RSSD. Otherwise, you can restart Replication Server to ensure the proper execution of your derived function-string class.

After you update `rs_translation`, you can use the derived function-string class for a connection to the corresponding non-ASE database.

Object Publication and Subscriptions Limitations

There are limitations to object publications and subscriptions in a Sybase replication system.

The limitations are:

- 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. See *Datatype Translation and Mapping* on page 229.

- 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). See the *Replication Server Administration Guide: Volume 1 > Manage Replicated Tables > Translate Datatypes Using HDS*.

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.

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.

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.

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 Log Transfer Language (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.

The implications of the Replication Agent connections:

- 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.

Support for Direct-Load Materialization

Some Replication Agents facilitate direct load materialization in Replication Server by providing the connectivity that Replication Server needs to query the primary database.

See *Direct Load Materialization* on page 257. To determine if direct-load materialization is supported for your primary database, see the appropriate chapter in this document for your primary database.

Non-ASE Replication Agents

Sybase offers non-ASE Replication Agents such as Replication Agent for DB2 UDB and Replication Agent.

Replication Agent for DB2 UDB

Replication Agent for DB2 UDB provides primary data server support for a DB2 UDB server running on IBM z/OS platforms.

Replication Agent for DB2 UDB 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.

Enterprise Connect Data Access

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 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

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

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.

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 a file.

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, see 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**.

To use direct-load materialization, you must have two database connections: one for incoming data and one for outgoing data. You must create the connection for incoming data—which is used by Replication Agent—using the appropriate connection profile (rs_rs_to_*<replicate>*_ra, where *<replicate>* is oracle, mssql, udb, or hanadb). Use a connection profile supporting normal replication to create the connection for outgoing data.

ECDA Database Gateways

ECDA database gateway applies transactions from a Replication Server to a non-ASE replicate database in a Sybase replication system.

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

  For an overview of the expectations of a replicate data server and gateway, see *Replication Server Design Guide > Data Replication into Non-Adaptive Server Data Servers*.
- Datatype representations must be translated to match the native datatypes of the replicate database. Replication Server provides 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 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.

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.

ECO has these advantages over ECDA:

- 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.

ExpressConnect for HANA DB

ExpressConnect for HANA DB is a library loaded by Replication Server for replication to HANA DB.

ExpressConnect for HANA DB:

- Runs as part of Replication Server
- Uses the HANA DB ODBC driver
- Supports both SAP® Secure Store and standard authentication
- Does not require text pointers for LOB replication and does not use custom function strings that manipulate text pointers

IBM DB2 for z/OS as Primary Data Server

You must consider 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.

When using Replication Agent for DB2 UDB, consider the following:

- 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 > Replication Server Commands >* ***sysadmin issue_ticket***.

DB2 UDB Primary Database Permissions

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**.

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.

Primary Data Server Connectivity

The Replication Agent for DB2 UDB requires a valid user ID that is defined to IBM z/OS and granted execute permission to the correct DB2 UDB plan and package to connect to a primary DB2 UDB data server in an IBM z/OS environment.

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.

Replication Server System Database Connectivity

Replication Agent for DB2 UDB does not require access to the Replication Server System Database (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_repdef**=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

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**.)

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.

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 *Replication Agent for DB2 UDB User and Troubleshooting Guide*

> Replication Agent Setup > DB2 Source Table Considerations > DB2 Table Names and Reserved Keywords.

DB2 for z/OS Primary Datatype Translation

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 for 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 *Replication Agent for DB2 UDB Installation Guide > LTM for MVS Configuration Parameters*.

Materialization

Materialization is the process of initially populating the replicate database with a copy of the data from the primary database.

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 *Replication Agent for DB2 UDB User and Troubleshooting Guide > Replication Server Setup > Task 3: Materializing Replicate Tables > Using Replication Agent Materialization*.

IBM DB2 for Linux, UNIX, and Windows as Primary Data Server

Learn about 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. See the *Replication Agent Administration Guide* and the *Replication Agent Release Bulletin*.

DB2 UDB System Management

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 may be affected by the transaction log.

- 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.

DB2 UDB Primary Database Permissions and Limitations

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.

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

Primary Data Server Connectivity

Replication Agent for UDB requires some tasks to perform to connect to a primary DB2 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 *Replication Agent Installation Guide > Installing Sybase Replication Agent > Setting Up Connectivity to the Primary Database*.

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

Replication Server and RSSD Connectivity

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 `interfaces` 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 *Replication Agent Installation Guide > Preparing for Installation*.

Replication Agent Objects

When you initialize Replication Agent using **pdb_xlog init**, it creates objects that support replication in the primary database.

See the *Replication Agent Primary Database Guide* for details.

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. Java Procedures for Truncation table 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.

Java Procedures for Truncation

Lists the Java procedures that are created during the Replication Agent initialization and used in log 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

Find the name of the Replication Agent database objects generated by Replication Agent instance.

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

Consider additional issues specific to heterogeneous replication.

All the installation issues and configuration parameter details for a primary DB2 UDB data server are in the *Replication Agent for DB2 UDB Installation Guide*.

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.

Java Heap Size

Make sure the size of the JVM heap is large enough for your primary DB2 UDB data server.

Set the DB2 UDB **java_heap_sz** configuration parameter to 2048 or a larger value.

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 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.

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

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 mappings, see the *Replication Agent Primary Database Guide > Replication Agent for UDB > DB2 Universal Database-Specific Considerations > DB2 UDB Datatype Compatibility*.

Microsoft SQL Server as Primary Data Server

Consider primary database issues 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

sybfilter driver is use to make the Microsoft SQL Server log files readable before Replication Agent can replicate data.

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. See the *Replication Agent Primary Database Guide*.

Microsoft SQL Server System Management

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

The user ID that the Replication Agent instance uses to log in to Microsoft SQL Server must have access to the primary database.

Replication Agent for Microsoft SQL Server creates database objects to assist with replication tasks in 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. See the *Replication Agent Release Bulletin* for the latest version of the Microsoft SQL Server JDBC.

Setting the CLASSPATH Environment Variable

Learn to set 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 the *Replication Agent Installation Guide > Preparing for Installation*.

Replication Server and RSSD Connectivity

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 `interfaces` 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 *Replication Agent Installation Guide > Preparing for Installation*.

Replication Agent Objects

Replication Agent creates objects in the 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.*

There are two variables in 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. 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, Procedures, Marker, and Trigger Objects

The table and procedure objects, marker procedures and marker shadow tables that are considered Replication Agent objects, as well as commands that are considered Replication Agent trigger objects are listed in the Replication Agent Primary Database Guide.

Insert and delete permissions are granted to Public only on the DDL shadow table for the database name prefixddl_trig_xxx. No permissions are granted on other tables.

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.

Microsoft SQL Server Primary Database Configuration

Learn about the additional issues that are specific to heterogeneous replication.

All the installation issues and configuration parameter details for a Microsoft SQL Server primary data server are in the *Replication Agent Installation Guide*.

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

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

Replication Definitions for Primary Tables in Microsoft SQL Server

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

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.

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 the *Replication Server Administration Guide Volume 1 > Manage Replicated Tables > Modify Replication Definitions > Altering Replication Definitions > Replication Definition Change Request Process*.

Microsoft SQL Server Primary Datatype Translation

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.

For a complete list of the Microsoft SQL Server datatype mappings, see the *Replication Agent Primary Database Guide > Replication Agent for Microsoft SQL Server> Microsoft SQL Server-Specific Considerations> Microsoft SQL Server Datatype Compatibility*.

Oracle as Primary Data Server

Learn about 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. See the *Replication Agent Administration Guide* and the *Replication Agent Release Bulletin* for more information.

Replication Definitions for Primary Tables in Oracle

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

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.

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 the *Replication Server Administration Guide Volume 1 > Manage Replicated Tables > Modify Replication Definitions > Altering Replication Definitions > Replication Definition Change Request Process*.

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

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 may be affected if a Sybase replication system is incorporated.

While the Replication Agent accesses 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

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, see the *Replication Agent Primary Database Guide*.

Primary Data Server Connectivity

Replication Agent requires a JDBC driver to connect to an Oracle primary database.

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 the *Replication Agent Installation Guide > Preparing for Installation*.

Replication Server and RSSD Connectivity

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 `interfaces` 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 *Replication Agent Installation Guide > Preparing for Installation*.

Replication Agent Objects

Replication Agent creates objects in the primary database to assist with replication tasks.

There are two variables in 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 nonalphanumeric characters allowed in the prefix string specified by **pdb_xlog_prefix**. This list of allowed characters is database-specific. For example, 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 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.

Oracle Primary Database Configuration

Learn about the additional issues specific to heterogeneous replication.

All the installation issues and configuration parameter details for an Oracle primary data server are provided in the *Replication Agent Installation Guide*.

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 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 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

Datatype translation and mapping provides a complete list of datatype mapping for Oracle datatypes.

See also

- *Datatype Translation and Mapping* on page 229

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*.

IBM DB2 for z/OS as Replicate Data Server

Learn about the primary database issues and considerations specific to the DB2 UDB for z/OS data server in a Sybase replication system.

For information about basic replication system administration, see the *Replication Server Administration Guide Volume 1*.

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

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.

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. Do not use this parameter with Replication Server version 12.0 or later. The default setting should be " "(blank space).

Note: This system management issue is specific to a replicate DB2 UDB for z/OS data server.

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.

The tables include:

- `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

Replication Server requires maintenance user ID that you specify in the Replication Server **create connection** command to apply transactions in a replicate database.

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

Replication of large object (LOB) datatypes (`BLOB` and `CLOB`) is supported directly by MainframeConnect DirectConnect for z/OS Option.

Additionally, 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

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

See also

- *Class-Level Datatype Translations to RSSD* on page 77
- *Objects in the DB2 UDB for z/OS and Connection Properties* on page 77
- *ECDA Settings* on page 78
- *Dynamic SQL Settings* on page 78
- *Command Batching Settings* on page 78

Replication Server Installation

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

Function Strings, Error Classes, and User Defined Datatypes

Function strings are added to the Replication Server default **rs_db2_function_class**.

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.

Note: ExpressConnect for Oracle and ExpressConnect for HANA DB do not use LOB pointers to manage LOB data. Consequently, the Replication Server system functions used to managing LOB pointers are unavailable to ExpressConnect for Oracle and ExpressConnect for HANA DB. These functions–which include **rs_get_textptr**, **rs_textptr_init**, and **rs_writetext**–are visible to ExpressConnect, but their use is ignored by Replication Server.

Connection Profiles

Connection profiles allow you to configure your connection with a predefined set of properties that match your primary and replicate database replication requirements.

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. See the *Replication Server Reference Manual > Replication Server Commands >* ***create connection*** for a complete list of the other options for **create connection** command.

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**.

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` – installs Adaptive Server-to-DB2 UDB class-level translations.
- `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.

To see all the available profiles, use the **admin show_connection_profiles** command.

Objects in the DB2 UDB for z/OS and Connection Properties

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

They also set these connection properties:

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

Additional Settings

Learn about the additional settings provided to support replication.

The settings include:

- ECDA settings
- Dynamic SQL settings
- Command Batching settings

ECDA Settings

Learn about the values for the properties of the ECDA and DirectConnect access service configuration files.

Use the following settings in the ECDA configuration file:

```
TransactionMode=long
Allocate=connect
SQLTransformation=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
```

Dynamic SQL Settings

Dynamic SQL is supported in ECDA 12.6.1 and later.

Command Batching Settings

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
```

See the *Replication Server Reference Manual > Replication Server Commands >* ***alter connection*** for information on setting **batch** and **dsi_cmd_separator** options by using the **alter connection** command.

IBM DB2 for Linux, UNIX, and Windows as Replicate Data Server

Learn about the primary database issues and considerations specific to the DB2 UDB data server for Linux, UNIX, and Windows in a Sybase replication system.

For information about basic replication system administration, see the *Replication Server Administration Guide Volume 1*.

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 three tables in the replicate database to support Replication Server operations

The tables include:

- `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.

DB2 UDB Replicate Database Permissions and Limitations

Replication Server requires a maintenance user ID that you specify using the Replication Server **create connection** command to apply transactions in a replicate database.

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

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.

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 `interfaces` 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 `interfaces` file to identify the host and port where the ECDA Option for ODBC database gateway server is listening. The

`interfaces` file entry name must match the **server_name** portion of the Replication Server database connection.

DB2 UDB 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 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)

See also

- *Class-Level Datatype Translations to RSSD* on page 84
- *Objects in the DB2 UDB Replicate Database and Connection Properties* on page 85

Replication Server Installation

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

Function Strings, Error Classes, and User Defined Datatypes

Function strings are added to the Replication Server default **rs_udb_function_class**.

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.

To find the error action defined for an error class, see the *Replication Server Reference Manual > RSSD Stored Procedures >* ***rs_helperror***.

Connection Profiles

Connection profiles allow you to configure your connection with a predefined set of properties that match your primary and replicate database replication requirements.

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. See the *Replication Server Reference Manual > Replication Server Commands >* ***create connection*** for a complete list of the other options for **create connection** command.

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**.

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` – installs Adaptive Server-to-DB2 UDB class-level translations.
- `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.

To see all the available profiles, use the **admin show_connection_profiles** command.

Objects in the DB2 UDB Replicate Database and Connection Properties

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

The connection profiles set these connection properties:

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

Additional Settings

Learn about the additional settings provided to support replication.

The settings include:

- 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, see the *Replication Server Reference Manual > alter connection*.

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 *Replication Server Administration Guide Volume 2 > Performance Tuning*.

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

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()))
```

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())
```

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.

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 of 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.

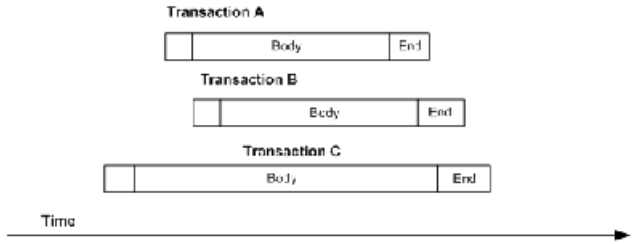
no_wait

The **no_wait** method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit.

no_wait 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 the "Thread Timing with wait_for_commit Serialization Method" diagram.

Note: You can only set **dsi_serialization_method** to **no_wait** if **dsi_commit_control** is set to "on".

Figure 3: Thread Timing with the no_wait Serialization Method



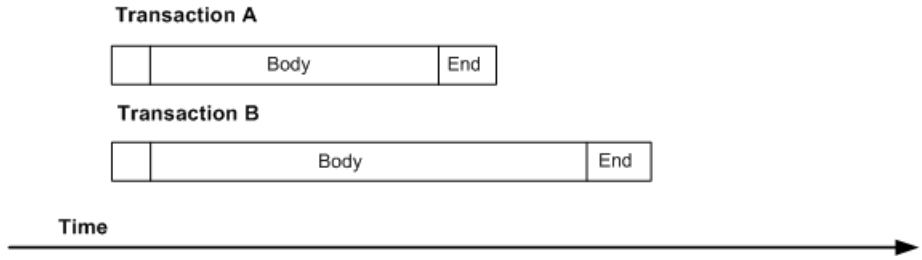
no_wait provides the better opportunity for increased performance, but also provides the greater risk of creating contentions.

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.

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

Figure 4: Thread Timing with wait_for_start Serialization Method

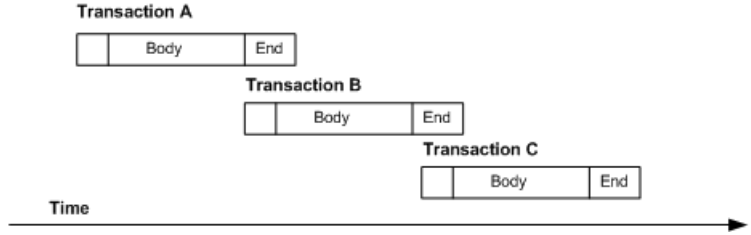


wait_for_commit

In **wait_for_commit** 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 the figure.

Figure 5: Thread Timing with wait_for_commit Serialization Method

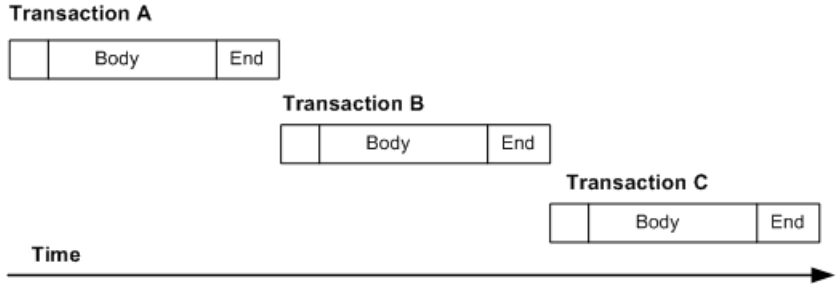


This method maintains transaction serialization by instructing the DSI to wait until a transaction is ready to commit before initiating the next transaction. The next transaction can be submitted to the replicate data server while the first transaction is committing, since the first transaction already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

Figure 6: Thread Timing with wait_after_commit Serialization Method



Microsoft SQL Server as Replicate Data Server

Learn about 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

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

Replication Server requires a maintenance user ID that you specify using the Replication Server **create connection** command to apply transactions in a replicate database.

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 `interfaces` 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 `interfaces` file to identify the host and port where the ECDA Option for ODBC database gateway server is listening. The `interfaces` 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

See also

- *Class-Level Datatype Translations to RSSD* on page 95
- *Objects in the Microsoft SQL Server Database and Connection Properties* on page 96

Replication Server Installation

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

Function Strings, Error Classes, and User Defined Datatypes

Function strings are added to the Replication Server default **rs_msss_function_class**.

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.

To find the error action defined for an error class, see *Replication Server Reference Manual > RSSD Stored Procedures >* ***rs_helperror***.

Connection Profiles

Connection profiles allow you to configure your connection with a predefined set of properties that match your primary and replicate database replication requirements.

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. See the *Replication Server Reference Manual > Replication Server Commands >* ***create connection*** for a complete list of the other options for **create connection** command.

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**.

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` – installs Adaptive Server-to-Microsoft SQL Server class-level translations.
- `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.

To see all the available profiles, use the **admin show_connection_profiles** command.

Objects in the Microsoft SQL Server Database and Connection Properties

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

The connection profiles set these connection properties:

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

Additional Settings

Learn about the additional settings provided to support replication.

The settings include:

- 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, see the *Replication Server Reference Manual > alter connection*.

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 *Replication Server Administration Guide Volume 2 > Performance Tuning*.

External and Internal 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 *Replication Server Administration Guide Volume 2 > Performance Tuning*.

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.

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 of 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.

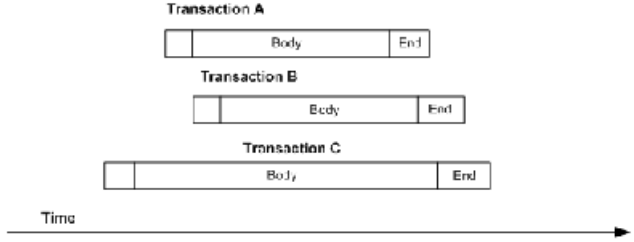
no_wait

The **no_wait** method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit.

no_wait 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 the "Thread Timing with wait_for_commit Serialization Method" diagram.

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

Figure 7: Thread Timing with the no_wait Serialization Method



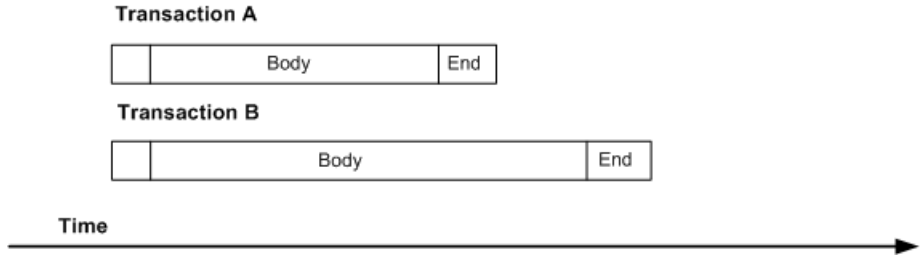
no_wait provides the better opportunity for increased performance, but also provides the greater risk of creating contentions.

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.

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

Figure 8: Thread Timing with wait_for_start Serialization Method

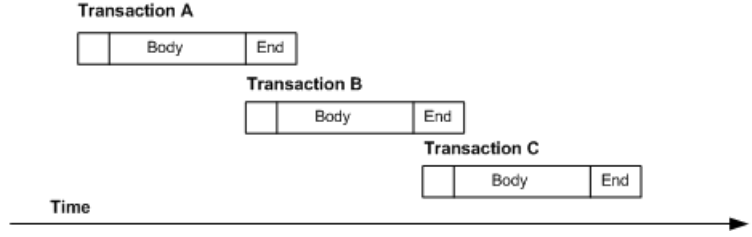


wait_for_commit

In **wait_for_commit** 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 the figure.

Figure 9: Thread Timing with wait_for_commit Serialization Method

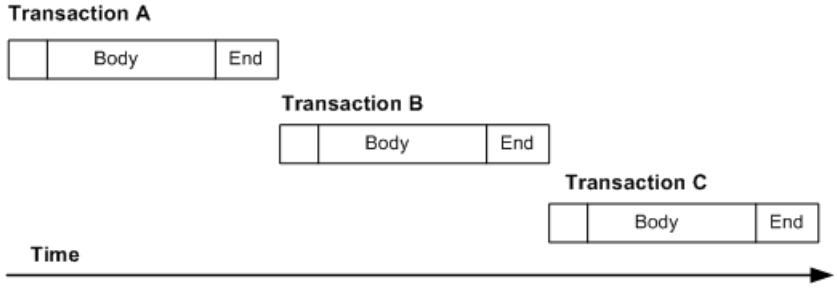


This method maintains transaction serialization by instructing the DSI to wait until a transaction is ready to commit before initiating the next transaction. The next transaction can be submitted to the replicate data server while the first transaction is committing, since the first transaction already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

Figure 10: Thread Timing with wait_after_commit Serialization Method



Oracle as Replicate Data Server

Learn about the replicate database issues and considerations specific to the Oracle data server in a Sybase replication system.

Oracle Replicate Data Servers

You replicate to an Oracle data server, using ExpressConnect for Oracle.

ExpressConnect for Oracle 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.

The created tables include:

- `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

Replication Server requires a maintenance user ID that you specify using the Replication Server **create connection** command to apply transactions in a replicate database.

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.
- **EXECUTE** authority on all replicate stored procedures.

Replicate Database Connectivity for Oracle

Replication Server connects to an Oracle replicate database ExpressConnect for Oracle (ECO).

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.

Specifying How Replication Server Replicates Stored Procedures
Set **dsi_proc_as_rpc** on if you use ExpressConnect for Oracle. ECO only supports stored procedure replication using remote procedure calls (RPC). By default, Replication Server sets **dsi_proc_as_rpc** on if you use one of the Oracle ECO connection profiles when you create the connection to the Oracle database from Replication Server. See *Replication Server Options 15.5 > Installation and Configuration Guide ExpressConnect for Oracle 15.5 > Configuring ExpressConnect for Oracle*.

Oracle 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 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

    You can connect using ExpressConnect for Oracle. When using ExpressConnect for Oracle, the version or option name of the connection profile should be “eco”.
- Additional settings:
  -
  - Settings for Command Batching
  - Settings for Dynamic SQL

See also

- *Class-Level Datatype Translations to RSSD* on page 107
- *Objects in the Oracle Replicate Database and Connection Properties* on page 107
- *Command Batching Settings* on page 108
- *Dynamic SQL Settings* on page 110

Replication Server Installation

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

Function Strings, Error Classes, and User Defined Datatypes

Function strings are added to the Replication Server default `rs_oracle_function_class`.

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.

Warning! ExpressConnect for Oracle does not support the use of custom function strings for text and image processing.

Connection Profiles

Connection profiles allow you to configure your connection with a predefined set of properties that match your primary and replicate database replication requirements.

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. See the *Replication Server Reference Manual > Replication Server Commands >* ***create connection*** for a complete list of the other options for **create connection** command.

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**.

Class-Level Datatype Translations to RSSD

Class-level translations identify primary datatypes and the replicate datatypes the data must 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` – installs Adaptive Server-to-Oracle class-level translations.
- `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.

To see all the available profiles, use the **admin show_connection_profiles** command.

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
```

Objects in the Oracle Replicate Database and Connection Properties

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.

The connection profiles set these connection properties:

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

Additional Settings

Learn about the additional settings provided to support replication.

The settings include:

- ExpressConnect settings
- Command Batching settings
- Trigger Firing settings
- Oracle Flashback settings
- Dynamic SQL settings

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;;"

Warning! ExpressConnect for Oracle does not support the use of custom function strings for text and image processing.

Array Processing in ExpressConnect for Oracle

Ensure that array processing in ExpressConnect for Oracle is applied only to tables that have a table-level replication definition. The performance enhancement provided by array processing within the ExpressConnect for Oracle connection requires the information included in the table-level replication definition.

Command Batching Settings

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_separator 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. See the *Replication Server Administration Guide Volume 2 > Command Batching for Non-ASE Servers*.

Trigger Firing Settings

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.

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.

Controlling Trigger Firing

Control trigger firing through **RS_TRIGGER_CONTROL** package, which is automatically installed when a connection to the replicate Oracle database is created through connection profiles.

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*.

Oracle Flashback Settings

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.

Dynamic SQL Settings

The use of Dynamic SQL for Oracle is supported with Replication Server using ExpressConnect.

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 using either:

- Replication Server internal tables and function strings, or
- The `rs_threads` system table in the replicate database.

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 *Replication Server Administration Guide Volume 2 > Performance Tuning*.

External and Internal Commit Control

Replication Server does not support external commit control when Oracle is the replicate database.

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.

For Oracle, you should only use **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_after_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_after_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 of a commit or a rollback.

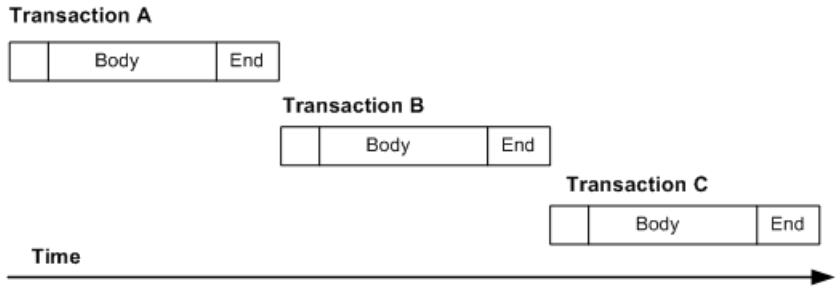
While providing commit consistency, the serialization method defines whether the beginning of the transaction waits for the previous transaction to become ready to commit or if the beginning of the transaction can be processed earlier.

wait_after_commit

wait_after_commit specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it has committed completely.

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.

Figure 11: Thread Timing with wait_after_commit Serialization Method



Sybase IQ as Replicate Data Server

Learn about the replicate database issues and considerations specific to the Sybase IQ data server in a Sybase replication system and how to connect and configure replication to Sybase IQ.

Sybase IQ is the ideal platform for reporting and data analysis. However, to be more effective for reporting, Sybase IQ requires real time data.

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 by performing compilation and bulk apply.

Real-Time Loading Solution

RTL groups as many compilable transactions as possible, compiles the transactions in the group into a net change, and then uses the bulk interface in the replicate database to apply the net change to the replicate database.

When replicating into Sybase IQ replicate databases, RTL uses:

- Compilation – rearranges replicate data by each table, and each **insert**, **update**, and **delete** operation, and compiling the operations into net-row operations.
- Bulk apply – applies the net result of the compilation operations in bulk using the most efficient bulk interface for the net result. Replication Server uses an in-memory net-change database to store the net row changes, which it then applies to the replicate database.

RTL improves performance for replication to Sybase IQ compared to the continuous replication mode and a staging solution for example, by using:

- Reduced number of external components – reduced maintenance costs and overhead, since there is no requirement for the staging database.
- Reduced latency – no overhead from the staging solution and with replication directly into Sybase IQ.
- Improved usability – the RTL configuration does not require any of: function-string mapping, DSI suspend and resume, data population from staging database to Sybase IQ, scheduling activities for the staging solution.
- Compilation and bulk apply – instead of sending every logged operation, compilation removes the intermediate **insert**, **update**, or **delete** 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 that Replication Server sends a smaller number of commands to Sybase IQ to process.

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.

As Replication Server compiles and combines a larger number of transactions into a group, bulk operation processing improves; therefore, replication throughput and performance also improves. You can adjust group sizes to control the amount of data that is grouped together for bulk apply.

License

Replication to Sybase IQ using real-time loading is available in the Real-Time Loading Edition product edition. See *Replication Server Installation Guide > Planning Your Installation > Obtaining a License*.

Database and Platform Support

- Sybase IQ – you can use real-time loading to replicate into Sybase IQ version 12.7 ESD #3 and later. See *Replication Server Release Bulletin > Product Compatibility > Replication Server Interoperability* for the latest supported Sybase IQ versions and platforms.
- Adaptive Server – Replication Server supports replication to Sybase IQ from Adaptive Server version 15.0.3 or version 15.5 and later.
- Oracle – Replication Server supports replication to Sybase IQ from Oracle 10g and 11g. See *Replication Server Options 15.5 > Release Bulletin Replication Agent 15.5 > Product Summary > Compatible Products*.

64-bit Support

You can achieve optimal performance using 64-bit hardware platforms. See *Replication Server New Features Guide > New Features in Replication Server Version 15.5 > Support for 64-bit Computing Platforms*.

RTL Compilation and Bulk Apply

During compilation, RTL rearranges data 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.

If a replicate table contains multiple unique keys, the primary key in the table replication definition must contain all columns which are named in the unique indexes. Otherwise, replication may produce duplicate key errors.

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** results in no reduction.
- An **update** followed by a **delete** results in a **delete**.
- An **insert** followed by an **update** results in an **insert** where the two operations are reduced to a final single operation that 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 that 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. In this example, `T` is a table created earlier by the command: **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. 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 uses an **insert**, **delete**, and **update** table in an in-memory net-change database to store the net-row changes it applies to the replicate database. Net-row changes are sorted by replicate table and by type of operation—**insert**, **update**, or **delete**—and are then ready for bulk interface.

RTL directly loads **insert** operations into the replicate table. Since Sybase IQ does not support bulk **update** and **delete**, RTL loads **update** and **delete** operations into temporary worktables 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 worktables are created and dropped dynamically.

In Example 2, where compilation results in `update T set c = 16 where k = 1`:

1. RTL creates the *#rs_uT(k int, c int)* worktable.
2. RTL performs an **insert** into the worktable:
   ```
   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 configuration parameters.

There is no data loss, although RTL does not apply row changes in the same order in which the changes are logged:

- For different data rows, the order in which row changes are applied does not affect the result.
- In the same row, applying **delete** before **insert** after compilation maintains consistency.

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. 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.

See also

- *Net-Change Database Size* on page 128

Monitoring the Net-Change Database

Access net-change database instances and monitor a net-change database.

Use the **sysadmin cdb** command to monitor a net-change database.

See *Replication Server Reference Manual > Replication Server Commands > **sysadmin cdb***.

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:

- **Insert** triggers do not fire, as the RTL process performs a bulk load of net new rows directly into the table. **Update** and **delete** triggers continue to fire when Replication Server applies the net results of compilation to the replicate database. However, row modifications that Replication Server compiles, and that are no longer in the net results, are invisible to the triggers. Triggers can detect only the final row images.

  Suppose you use Replication Server to audit user updates using a `last_update_user` column in a table schema with a trigger logic that associates a user to any column in the table modified by the user. If userA modifies `colA` and `colC` in the table and then userB modifies `colB` and `colD`, when the trigger fires, the trigger logic can detect only the last user who modified the table, and therefore the trigger logic associates userB as the user that modified all four columns. If you define triggers that contain similar logic where every individual row modification must be detected, you may have to 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 constraints in replication definitions. To avoid constraint errors, RTL loads tables according to replication definitions.
- RTL does not support any parallel DSI serialization methods, except for the default **wait_for_commit** method.
- RTL does not support customized function strings and treats customized function strings as noncompilable commands.
- Replication Server reverts to log-order, row-by-row continuous replication when it encounters:
  - Noncompilable commands – stored procedures, SQL statements, system transactions, and Replication Server internal markers, and **rs_ticket**.
  - Noncompilable transactions – a transaction that contains noncompilable commands.
  - Noncompilable tables – tables with RTL disabled, with customized function strings, and with referential constraint relationships with tables that RTL cannot compile.
  - Runtime noncompilable tables - this occurs when a transaction contains minimally packed updates, for example when using the **replicate minimal columns** clause in the replication definition for that table, and when the transaction modifies the primary key value.
- Replication Server does not support RTL for Sybase IQ views. Set **dsi_compile_enable** off for the view to mark the view noncompilable.
- For tables without primary keys where there are no table replication definitions, Replication Server converts updates to the table to primary-key updates as Replication Server treats all columns, except `text` or `image` columns, as primary keys.
- RTL ignores parameters such as **dsi_partition_rule** that can stop transaction grouping.
- If errors occur during RTL processing, Replication Server retries compilation with progressively smaller transaction groups until it identifies the transaction that failed compilation, then applies the transaction using continuous replication.

- To realize performance benefits, 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**.
- RTL performs row-count validation to ensure replication integrity. 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 these Sybase IQ commands 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.
- When converting from microseconds in the primary data server to milliseconds on the Sybase IQ replicate data server, Replication Server pads the last three digits of the microsecond with zeros in the `time`, `datetime`, and `smalldatime` columns in the `TIMESTAMP` datatype causing replication to fail. See *Sybase IQ Reference: Building Blocks, Tables, and Procedures > Compatibility with Other Sybase Databases > Data Types > Date, Time, Datetime, and Timestamp Data Types > Compatibility of Datetime and Time Values from ASE* .

  When you manually materialize a replicate table using the Sybase IQ **INSERT ... LOCATION** statement, Sybase IQ fills in the last three digits of the microsecond in `TIMESTAMP` columns to use the values 000, 333, or 666. For example:

  ```
  insert test_datetime_iq4 location 'zeus.primaryDB4'
       { select c1,c2,c3,c4,c5 from test_datetime_iq4 }
  ```

  where c2 is `datetime` and c4 is `time`.

  To ensure that RTL continues to replicate to Sybase IQ after you use **INSERT ... LOCATION** to materialize a table, manually materialize the Sybase IQ tables by enforcing explicit conversion of the `TIMESTAMP` datatype:

  ```
  insert test_datetime_iq4 location 'zeus.primaryDB4'
        { select c1,convert(varchar,c2,109),
                 c3,convert(varchar,c4,20),c5
                 from test_datetime_iq4 }
  ```

- By default, Oracle performs minimal logging. Therefore, if you are using database replication definitions, either create table replication definitions or enable full logging to ensure the **update** command works correctly. If you choose to create table replication definitions, you can create the definitions in Replication Agent or Replication Server:
  - Replication Agent for Oracle – to automatically create replication definitions at Replication Server when one or more tables are marked for replication, either set **pdb_auto_create_repdefs** to **true** before you mark the table for replication or execute **rs_create_repdef** after you mark the table . See the *Replication Agent Reference Manual* in Replication Server Options.

- Replication Server – execute **create replication definition** with the **send standby** clause to create the replication definition directly in Replication Server. See the *Replication Server Reference Manual*.

See also

- *Tables with Referential Constraints* on page 135
- *RTL Performance Tuning* on page 125

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 the Sybase IQ replicate database:

- `rs_threads` – 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` – 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 most recent 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` – contains the execution results of Replication Server command **rs_ticket**. You can isssue 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 to compare the results from different rows. Manually truncate the data in `rs_ticket_history` table if necessary.

Worktables

RTL creates temporary worktables inside the IQ temporary store of the Sybase IQ database to support RTL bulk apply.The worktables 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 worktables, use the Sybase IQ **alter dbspace** command. See the Sybase IQ documentation for your version for more information. For example in Sybase IQ 15.0 and later:

```
ALTER DBSPACE dbspace-name ADD FILE FileHist3
'/History1/data/file3' SIZE 500MB
```

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 a data server name—**server_name**—and a database name—**db_name**. The replicate Replication Server looks for an `interfaces` 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 `interfaces` file to identify the host and port where the Sybase IQ replicate data server is listening. The `interfaces` 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 *Replication Server Configuration Guide > Configure Replication Server and Add Databases Using* **rs_init** *> Configuring a New Replication Server > Editing the Interfaces File*.

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.

In Sybase IQ 15.0 and later, enter: `set option public.STRING_RTRUNCATION = "OFF"` to avoid the "`Right truncation of string data`" error when Replication Server inserts data into Sybase IQ using **INSERT ... LOCATION**.

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.

Sybase IQ Replicate Database Permissions

To apply transactions in a replicate database, Replication Server and Sybase IQ require a maintenance user ID.

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 worktables 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.

Granting Authority to a Maintenance User ID

Grant DBA and RESOURCE authority if you are starting with a simple setup or are testing replication to Sybase IQ.

1. Use the Sybase IQ `rssetup.sql` sample script to create the maintenance user for Sybase IQ with relevant privileges.

   Warning! If there is already a maintenance user ID, the script resets 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
   ```

   This script is 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. Verify that the Sybase IQ database is compatible with Transact-SQL® (For IQ DBA).

   See *Sybase IQ Reference: Statements and Options > Database Options > Transact-SQL Compatibility Options* and *Sybase IQ Reference: Building Blocks, Tables, and Procedures > Compatibility with Other Sybase Databases*.
3. Grant the appropriate permissions to all tables and stored procedures that are to participate in replication.

Sybase IQ Replicate Database Configuration

Learn about the 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 predefined 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.

Connection profiles, such as **rs_ase_to_iq** and **rs_oracle_to_iq** are a part of the Replication Server installation package, and are registered when you install Replication Server. A 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 connection profile:
  - **rs_ase_to_iq** – installs Adaptive Server-to-Sybase IQ class-level translations.
  - **rs_oracle_to_iq** – translates Oracle datatypes to Sybase IQ datatypes

  To see all the available profiles, use the **admin show_connection_profiles** command.
- Creates the `rs_threads`, `rs_lastcommit`, and `rs_ticket_history` tables in the Sybase IQ replicate database.

- Sets the default function-string class and error class connection properties 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

Set up the connection to the replicate Sybase IQ database.

1. Use **create connection** with the **using profile** clause and the relavant connection profile, and specify your replicate Sybase IQ data server and database.
   For example to create a connection from an Oracle primary data server:

```
create connection to IQSRVR.iqdb
using profile rs_oracle_to_iq;standard
set username to dbmaint
set password to dbmaint
go
```

   You can create multiple replication paths to the Sybase IQ database to distribute replication loads. Use a unique maintenance user ID for each path.
2. Use **admin who** to verify that Replication Server connects successfully to Sybase IQ.

See also

- *Multi-Path Replication to Sybase IQ* on page 131

Setting Sybase IQ Database Options

You can use the **rs_session_setting** function with the **create function string** command to set the values for Sybase IQ parameters for the duration of the connection to the Sybase IQ replicate database. For example, you can set parameter values to optimize performance.

1. Create a new function-string class named **my_iq_fclass** and set **rs_iq_function_class** as the parent class:

```
create function string class my_iq_fclass
set parent to rs_iq_function_class
go
```

2. Create the **rs_session_setting** function string for the **my_iq_fclass** function-string class, and include the Sybase IQ parameters and values you want to set.
   For example, you can set the values of the **LOAD_MEMORY_MB**, **MINIMIZE_STORAGE**, and **JOIN_PREFERENCE** Sybase IQ database options to optimize performance:

```
create function string rs_session_setting
for my_iq_fclass
output language
'set temporary option Load_Memory_MB=''200''
set temporary option Minimize_Storage=''on''
set temporary option join_preference=5'
go
```

See *Replication Server Reference Manual > Replication Server System Functions > **rs_session_setting***.

3. Alter the connection to the `iqdb` database in the `IQSRVR` data server to use the **my_iq_fclass** function-string class:

```
alter connection to IQSRVR.iqdb
set function string class to my_iq_fclass
go
```

Enable RTL

After you have granted the relevant permissions and connected to the replicate Sybase IQ database, 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** off, Replication Server uses continuous log-order, row-by-row replication mode. For example, set **dsi_compile_enable** off for an affected table if replicating net-row changes causes problems, such as when there is a trigger on the table that requires all operations on the table to be replicated in log order, and therefore compilation is not allowed.

When you set **dsi_compile_enable** on, Replication Server disables **dsi_cmd_prefetch** and **dsi_num_large_xact_threads**.

Remember: You must set **dsi_bulk_copy** and **dynamic_sql** to **off** before you enable real-time loading (RTL) replication to Sybase IQ.

To enable and configure RTL at the database level to affect only the specified database, enter:

```
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.

RTL Performance Tuning

Replication Server automatically sets the recommended default values of several parameters. You can change the values of these parameters to tune replication performance.

You must execute a separate **alter connection** command for each parameter you want to change. Do not enter more than one parameter after entering **alter connection**.

See *Replication Server Reference Manual > Replication Server Commands >* ***alter connection*** for full descriptions of the parameters.

dsi_bulk_threshold

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.

Default is 20 net row change commands.

Example:

```
alter connection to IQSRVR.iqdb
set dsi_bulk_threshold to ‘15’
go
```

dsi_cdb_max_size

dsi_cdb_max_size specifies, in megabytes (MB), the maximum size of a net-change database that Replication Server can generate during RTL processing.

Default is 1024MB.

Example:

```
alter connection to IQSRVR.iqdb
set dsi_cdb_max_size to ‘2048’
go
```

Replication Server uses full incremental compilation for real-time loading to Sybase IQ. With full incremental compilation, if the number of commands in the compiled transaction segment within a net-change database instance exceeds the **dsi_compile_max_cmds** threshold, or if the net-change database instance size exceeds the **dsi_cdb_max_size** threshold, Replication Server instructs the net-change database instance to send its transaction to the replicate database and release the memory that the instance consumed.

dsi_compile_max_cmds

dsi_compile_max_cmds specifies, in number of commands, the maximum size of a group of transactions and commands that Replication Server can compile into one compiled transaction. When RTL reaches the maximum group size for the current group that it is compiling, RTL starts a new group. Replication Server creates a net-change database instance to store the compiled transaction. Replication Server increases the net-change database size to accommodate the maximum number of commands that **dsi_compile_max_cmds** allows for a group. When Replication Server reaches the maximum group size for the current group that it is compiling, Replication Server transfers the compiled transaction to the worktables in the replicate database, releases the memory consumed by that specific net-change database instance, starts a new group and creates a new net-change database instance for the 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.

Default is 10,000 commands.

Example:

```
alter connection to IQSRVR.iqdb
set dsi_compile_max_cmds to ‘50000’
go
```

dsi_compile_retry_threshold

dsi_compile_retry_threshold specifies a threshold value for the number of commands in a group. If the number of commands in a group containing failed transactions is smaller than the value of **dsi_compile_retry_threshold**, Replication Server does not retry processing the group in RTL mode, and saves processing time, thus improving performance. Instead, Replication Server switches to continuous replication mode for the group. Continuous replication mode sends each logged change to the replicate database according to the primary database log order.

Default is 100 commands.

You need not suspend and resume database connections when you set **dsi_compile_retry_threshold**. The parameter takes effect immediately after you execute the command.

Example:

```
alter connection to IQSRVR.iqdb
set dsi_compile_retry_threshold to '200'
go
```

See *Replication Server Administration Guide Volume 2 > Exceptions and Error Handling > Data Server Error Handling > Row Count Validation > Control Row Count Validation*.

dsi_command_convert

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 need not 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. If you disable row count validation by setting **dsi_row_count _validation** off, Sybase recommends that you set **dsi_command_convert** to **i2di,u2di** to avoid duplicate key errors and allow autosynchronization of databases during replication.
- **t2none** – do not replicate **truncate table**.

Default for **dsi_command_convert** is **none**, which means there is no command conversion.

Example:

```
alter connection to IQSRVR.iqdb
set dsi_command_convert to 'i2di,u2di'
go
```

See also

- *Memory Consumption Control* on page 128

Enhanced Retry Mechanism

The enhanced retry mechanism improves replication performance by reducing the number of times Replication Server retries compilation and bulk apply.

RTL attempts to group as many compilable transactions as possible together, compile the transactions in the group into a net change, and then use the bulk interface in the replicate database to apply the net changes to the replicate database. RTL invokes the retry mechanism when a replicate transaction resulting from RTL processing fails. If transactions in a group fail, RTL splits the group into two smaller groups of equal size, and retry the compilation and bulk application on each group. The retry mechanism identifies the failed transaction, allows Replication Server to execute error action mapping, and applies all transactions preceding the failed transaction in case DSI shuts down.

The net-change database in RTL acts as an in-memory repository for storing the net row changes of a transaction, that is, the compiled transaction. The content of the net-change database is an aggregation of commands from different primary transactions that RTL is not

applying in log order. Therefore, there is no means to identify a failed transaction without a retry mechanism. The retry mechanism splits a group and retries compilation and bulk application continuously as long as a transaction in the group fails. This continuous retry process can degrade performance.

The enhanced retry mechanism splits the group into three groups of equal size when RTL encounters a group containing transactions that fail, enabling the mechanism to more efficiently identify the group containing the failed transaction.

In addition, you can use the **dsi_compile_retry_threshold** parameter to specify a threshold value for the number of commands in a group. If the number of commands in a group containing failed transactions is smaller than the value of **dsi_compile_retry_threshold**, Replication Server does not retry processing the group in RTL mode, and saves processing time, thus improving performance. Instead, Replication Server switches to continuous replication mode for the group. Continuous replication mode sends each logged change to the replicate database according to the primary database log order.

Memory Consumption Control

RTL uses full incremental compilation to control memory consumption and you can control the net-change database size to reduce memory consumption.

SQT Memory Consumption Control for RTL

Control the maximum memory consumed by unpacked commands in the DSI SQT cache during transaction profiling in RTL .

The SQT thread monitors the memory consumed by commands unpacked by the RTL transaction profiling processes and referenced by the DSI SQT cache.

When Replication Server is replicating using RTL, the maximum amount of memory consumed by the DSI thead is the sum of **dsi_sqt_max_cache_size**, **sqt_max_prs_size** , and **dsi_cdb_max_size**. Setting **dsi_sqt_max_cache_size**, **sqt_max_prs_size** , and **dsi_cdb_max_size** smaller would reduce memory consumption but reduce replication performance. Tune your replication environment for optimum memory consumption and performance. See the *Replication Server Reference Manual > Replication Server Commands* to configure the parameters.

Net-Change Database Size

Reduce memory consumption by the net-change database by triggering the net-change database to flush data to the replicate database once the net-change database size reaches a threshold size.

Memory consumption refers to Replication Server data structures such as the net-change database, and the data that the structures store. Net-change databases are in-memory data structures. Net-change database memory consumption can increase drastically when Replication Server compiles commands applied on a table with a large number of columns, or tables with large `text` and `image` datatype values. For example, compiling 1,000,000 rows

in a table with 100 columns may consume approximately 10 times more memory than compiling the same number of rows in a table with 10 columns. Replication performance suffers when there is insufficient memory available for other processes and modules.

Replication Server uses the values you set for **dsi_cdb_max_size** and **dsi_compile_max_cmds** to control memory consumption. You can use **dsi_cdb_max_size** to control the maximum net-change database size that Replication Server can generate. Once the size reaches the threshold you set, Replication Server stops compiling new commands and transactions into the compiled transaction that Replication Server is building in the net-change database, performs the bulk apply of the compiled group to the replicate database, clears the net-change database, and releases the memory consumed by the net-change database.

The number of of net-change database instances that Replication Server generates depends on the values you set with **dsi_cdb_max_size** and **memory_limit**. The estimated memory requirements for a replication system using RTL is the number of replicate connections multiplied by **dsi_cdb_max_size**.

Full Incremental Compilation

Full incremental compilation improves replication performance for real-time loading (RTL) by reducing memory consumption during the processing of large compilable transactions that contain many commands.

Full incremental compilation can compile large transactions containing mixed **insert**, **delete**, or **update** operations. Replication Server uses full incremental compilation to apply a large compilable transaction to the replicate database, using multiple in-memory net-change database instances. Full incremental compilation divides a large transaction into a sequence of segments. Each segment consists of a group of commands.

Replication Server compiles each segment and creates a dedicated net-change database in which to to store one segment. Replication Server instructs the net-change database instance to send and apply the segment to the replicate database. Replication Server then closes the net-change database instance and releases the memory consumed. Replication Server creates another net-change database instance for the next transaction segment and continues to create and close net-change database instances in sequence for all the segments.

Therefore, instead of consuming a single large portion of memory for a large net-change database instance to hold a large transaction, full incremental compilation reduces the memory requirement to the memory consumed by a single smaller net-change database instance containing just a segment of the transaction. Full incremental compilation divides the memory requirement by the number of net-change database instances used. For example, when full incremental compilation applies a large transaction with 10 net-change database instances, the memory requirement is approximately one-tenth of the requirement without full incremental compilation.

Memory Control Parameters and Replication Server Processing

Replication Server actions depend on the values you set for memory control parameters.

Setting dsi_cdb_max_size to Different Values

Examples that show Replication Server applying a transaction with 100,000 updates on two tables. Table1 has 100 columns and requires approximately 4GB of memory, and Table2 has 10 columns requiring approximately one-tenth the memory—400MB.

dsi_cdb_max_size Value (MB)	Table Name	Impact on Replication Processing
1024 (default)	Table1	Prerequisite: Set **memory_limit** in Replication Server to a value large enough to allow the construction of 1GB net-change databases. Replication Server uses 4 1GB net-change database instances to apply the transaction.
1024 (default)	Table2	Prerequisite: Set **memory_limit** in Replication Server to a value large enough to allow the construction of 400MB net-change databases. Replication Server uses 1 400MB net-change database instance to apply the transaction.
4096	Table1	Prerequisite: Set **memory_limit** in Replication Server to a value large enough to allow the construction of 4GB net-change databases. Replication Server uses 1 4GB net-change database instance to apply the transaction.
4096	Table2	Prerequisite: Set **memory_limit** in Replication Server to a value large enough to allow the construction of 400MB net-change databases. Replication Server uses 1 400MB net-change database instance to apply the transaction.

Incremental Parsing for RTL

To further reduce memory consumption, enable incremental parsing by the DSI Scheduler thread by setting **dsi_incremental_parsing** on.

See *Incremental Parsing* in the *Replication Server Administration Guide Volume 2*.

Multi-Path Replication to Sybase IQ

Create multiple connections from Replication Server to the replicate Sybase IQ database to increase replication throughput and performance, and reduce latency and contention.

With multiple connections from an Adaptive Server or Oracle primary database to Replication Server and multiple connections from Replication Server to the replicate Sybase IQ database, you can create end-to-end multiple replication paths.

Database Support

- Primary database:
  - Adaptive Server 15.7 and later
  - Oracle 10g and 11g. See *Replication Server Options > Replication Agent Release Bulletin > Product Summary > Product Compatibility*.
- Replicate database – Sybase IQ version 15.1 and later. See *Replication Server Release Bulletin > Product Compatibility > Replication Server Interoperability* .

License

Multi-Path Replication™ is licensed as part of the Advanced Services Option. Replication to Sybase IQ using RTL is available in the Real-Time Loading Edition (RTLE). See *Replication Server Installation Guide > Planning Your Installation > Obtaining a License*.

See also

- *Heterogeneous Multi-Path Replication* on page 165
- *Multi-Path Replication from Adaptive Server to Sybase IQ* on page 179
- *Multi-Path Replication from Oracle to Sybase IQ* on page 195

Creating Alternate Replicate Connections to Sybase IQ

Use **create alternate connection** with the **using profile** clause to create an alternate connection from Replication Server to the replicate Sybase IQ database.

Prerequisites

Create the default connection to the replicate database before you create any alternate connections.

Task

You must specify the connection profile and connection profile version, and a unique maintenance user name for the default and each alternate connection.
Create an alternate connection to the Sybase IQ replicate database:

```
create alternate connection to dataserver.database
named conn_server.conn_db
```

```
using profile connection_profile;version
set username [to] user
set password [to] pwd
```

where:

- *dataserver* and *database* – are the replicate data server and database.
- *conn_server.conn_db* – the alternate replicate connection, which comprises the data server name and a connection name.
  - Each replicate connection name must be unique in a replication system.
  - If *conn_server* is different from *dataserver*, there must be an entry for *conn_server* in the interface file.
  - If *conn_server* is the same as *dataserver*, *conn_db* must be different from *database*.
- *connection_profile* – specifies the correct function-string class and error class for the replicate database, and additionally may contain class-level translation definitions and support for replicate database object creation:
  - **rs_ase_to_iq** – for Adaptive Server to Sybase IQ replication
  - **rs_oracle_to_iq** – for Oracle to Sybase IQ replication

  Note: You must specify the connection profile for each alternate connection you create to a Sybase IQ database.
- *version* – the connection profile version to use

  Note: You must specify the connection profile version for each alternate connection you create to a Sybase IQ database
- *user* – login name of the Replication Server maintenance user for each connection to the Sybase IQ database. Replication Server uses this login name to maintain replicated data. You must specify a user name if network-based security is not enabled.

  Note: You must use a different maintenance user name for each alternate connection you create to a Sybase IQ database. Ensure that you use unique maintenance user names even if you are creating connections from different Replication Servers to a Sybase IQ database. If you do not use unique user names, data duplication errors may occur. The replication system cannnot detect if you use the same user name to create connections from different Replication Servers.

For example, to create an alternate replicate connection named `IQSRVR.iqdb_conn2` to the `iqdb` replicate database in the `IQSRVR` Sybase IQ data server where the primary database is Adaptive Server and dbmaint2 is the maintenance user for `IQSRVR.iqdb_conn2`:

```
create alternate connection to IQSRVR.iqdb
named IQSRVR.iqdb_conn2
using profile rs_ase_to_iq;standard
set username to dbmaint2
set password to dbmaint2pwd
go
```

Altering or Dropping Alternate Replicate Sybase IQ Connections

Alter or drop default or alternate replicate connections to Sybase IQ by using the **alter connection** and **drop connection** commands.

The data server and database names that you specify in the commands can be the default or alternate replicate connection names.

You can use configuration parameters available to **alter connection** when you configure an alternate or default replicate connection.

For example, to set **dsi_bulk_threshold** to 15 for the `IQSRVR.iqdb_conn2` alternate replicate connection, enter:

```
alter connection to IQSRVR.iqdb_conn2
set dsi_bulk_threshold to '15'
go
```

Displaying Replicate Connection Information

Use the **replicate** parameter with **admin show_connections** to display information on all replicate connections.

For example, at the Replication Server controlling the replicate databases in the IQSRVR data server, enter:

```
admin show_connections, 'replicate'
```

You see:

```
Connection Name         Server       Database   User
-------------------     ----------   --------   ----------
IQSRVR.iqdb             IQSRVR       iqdb       db_maint
IQSRVR.iqdb_conn2       IQSRVR       iqdb       db_maint2
IQSRVR.iqdb_conn3       IQSRVR       iqdb       db_maint3
```

IQSRVR.iqdb is the default connection between the Replication Server and the `iqdb` database of the IQSRVR data server because the connection name matches the combination of the data server and database names.

IQSRVR.iqdb_conn2 and IQSRVR.iqdb_conn3 are alternate connections between the Replication Server and the `iqdb` database of the IQSRVR data server because the connection name does not match the combination of the data server and database names.

Replication Load Distribution

Use the distribution mode supported by the primary database to distribute the replication load over available replication paths.

Distribution Modes

With the Adaptive Server distribution by object binding mode, you can distribute objects such as tables and stored procedures over the multiple paths by binding an object to a specific path.

If you match the primary and replicate connection names, the object follows an end-to-end replication path from primary to replicate data server. See *Replication Server > Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Binding Objects to a Replication Path*.

With the Adaptive Server distribution by connection mode, the Adaptive Server RepAgent assigns transactions originated by different client processes to the available replication paths. Over time, the distribution of data tends to be balanced across all available paths. Replication performance improves and replication load distribution is more uniform if there are more RepAgent paths available and the number of client processes is large. See *Replication Server > Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Parallel Transaction Streams > Distribution Modes for Multi-Path Replication*.

Sybase IQ Multiplex Nodes

You can allocate replicate connections to different nodes in a Sybase IQ multiplex to distribute the replication load by creating connections to each node and creating the relevant `interfaces` file entries. See *Sybase IQ > Using Sybase IQ Multiplex*.

Setting Distribution Model

Set the distribution model for replication through multiple primary replication paths.

Prerequisites

Create the default and alternate connections from the primary Adaptive Server to Replication Server, and enable multithreaded RepAgent. Enable multiple scanners if you intend to set the distribution model to distibution by column filter.

Task

If you change the distribution model and you add new bindings or have existing bindings that RepAgent cannot associate with the new distribution model , RepAgent displays a warnimg that it will ignore some bindings under the new distribution model. However, RepAgent retains the inactive bindings. If you revert to the distibution model that corresponds with the type of the inactive bindings, RepAgent once again uses the formerly inactive bindings. For example, if you change to distribution by column filter from distribution by object binding, RepAgent ignores all the table and stored procedure bindings that you have set.

1. Set the distribution model:

   ```
   sp_config_rep_agent database, 'multipath distribution model',
   {'object' | 'connection' | 'filter'}
   ```

   where:

   - **multipath distribution model** is the distribution model parameter for **sp_config_rep_agent**
   - **object** – sets the model to distribution by object binding which is the default
   - **connection** – sets the model to distribution by connection

- **filter** – sets the model to distribution by filter

2. Quiesce Replication Server and restart RepAgent.

   See *Replication Server Administration Guide Volume 1 > Manage a Replication System > Quiesce Replication Server > Quiescing a Replication System*.

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.

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 a replication definition for each primary database. However, if multiple referential constraints conflict with each other, Replication Server randomly selects one.

See also

- *RTL Processing and Limitations* on page 116

Replication Definitions Creation and Alteration

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.

Display RTL Information

You can display information on configuration parameter properties and table references.

Display Configuration Parameter Properties

Use **admin config** to view information about database-level and table-level configuration parameters as shown in the examples.

- Database-level:
  - To display all database-level configuration parameters for the connection to the `nydb1` database of the NY_DS data server (`NY_DS.nydb1`), enter:

    ```
    admin config, "connection", NY_DS, nydb1
    ```
  - To verify that **dsi_compile_enable** is **on** for the connection to `NY_DS.nydb1`, enter:

    ```
    admin config, "connection", NY_DS, nydb1,dsi_compile_enable
    ```
  - To display all the database-level configuration parameters that have "enable" as part of the name, such as **dsi_compile_enable**, enter:

    ```
    admin config, "connection", NY_DS, nydb1,"enable"
    ```

Note: You must enclose "enable" in quotes because it is a reserved word in Replication Server. See *Replication Server Reference Manual > Topics > Reserved Words*.

- Table-level:

  To display all configuration parameters after using **dsi_command_convert** to set **d2none** on the `tb1` table in the `nydb1` database of the NY_DS data server, enter:

  ```
  admin config, "table", NY_DS, nydb1
  ```

See *Replication Server Reference Manual > Replication Server Commands > **admin config***.

Display Table References

Use **rs_helprep**, which you can execute on the Replication Server System Database (RSSD), to view information about table references and RTL information.

To display information about the **authors_repdef** replication definition created using **create replication definition**, enter:

```
rs_helprep authors_repdef
```

See *Replication Server Reference Manual > RSSD Stored Procedures > **rs_helprep***.

System Table Support in Replication Server

Replication Server uses the `rs_tbconfig` table to store support table-level configuration parameters, and the `ref_objowner` and `ref_objname` columns in the `rs_columns` table to support referential constraints.

See *Replication Server Reference Manual > Replication Server System Tables* for full table descriptions.

Mixed-Version Support and Backward Compatibility

RTL can replicate referential constraints specified in replication definitions only if the outbound route version is later than 15.5.

RTL works if the outbound 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.

Scenario for Replication to Sybase IQ

The scenario to set up replication to Sybase IQ using RTL and to test that replication works is described.

The Adaptive Server database administrator (ASE DBA) or the Oracle database administrator (Oracle DBA), the Sybase IQ database administrator (IQ DBA), and you, as the replication system administrator (RSA), must prepare Adaptive Server or Oracle, 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 or the ORA_DS primary Oracle 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.

See also

- *Multi-Path Replication from Adaptive Server to Sybase IQ* on page 179
- *Multi-Path Replication from Oracle to Sybase IQ* on page 195

Creating Interfaces File Entries

Create an entry in the `interfaces` files of the replicate Replication Server and the Sybase IQ data server for each other.

1. Create an entry for the replicate Replication Server in the `interfaces` file (`sql.ini` file in Windows) of the Sybase IQ data server.

   Note: Create an `interfaces` file for the Sybase IQ data server if the file is not in the $SYBASE directory (%SYBASE% directory in Windows) that Sybase IQ is using.

2. Create an entry for the Sybase IQ data server in the `interfaces` file of the replicate Replication Server.

   If you are creating connections to different Sybase IQ multiplex nodes, create entries for each of the affected nodes in the `interfaces` file of the replicate Replication Server.

Creating Test Tables

Create a test table in the primary and replicate databases, and grant maintenance user permissions to it to test that replication works.

1. In the primary database `pdb1` in the data server, create a table named `testtab` with three columns: `c1 integer`, `c2 integer` and `c3 char(10)`.
   For example, in Adaptive Server:

   ```
   use pdb1
   go
   create table dbo.testtab(c1 int primary key, c2 int,
   ```

```
c3 char(10))
go
```

See Oracle documentation to create a table in the Oracle database.

2. In the replicate database `iqdb` in the Sybase IQ IQSRVR data server, enter:

```
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
```

Creating the Connection to the Primary and Replicate Databases

Create the primary and replicate database connections.

1. Create the connection to the primary database.
   - Adaptive Server – use the Replication Server **rs_init** utility. See *Replication Server Configuration Guide > Configure Replication Server and Add Databases using* ***rs_init***.
   - Oracle – see the *Heterogeneous Replication Guide* and the Replication Server Options product documentation.
2. Create the connection to the Sybase IQ replicate database.

   See *Replication Server Reference Manual > Replication Server Commands >* ***create connection using profile*** .

   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.

   - Adaptive Server:

     ```
     create connection to IQSRVR.iqdb
     using profile rs_ase_to_iq;standard
     set username to dbmaint
     set password to dbmaint
     go
     ```

   - Oracle:

     ```
     create connection to IQSRVR.iqdb
     using profile rs_oracle_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.
   ```

3. 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
```

Enabling RTL

Enable RTL at the database level.

Prerequisites

Set **dsi_bulk_copy** and **dynamic_sql** to **off** before you enable real-time loading (RTL) replication to Sybase IQ.

Task

1. To enable and configure RTL at the database level to affect only the specified database, enter:

   ```
   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:

   ```
   suspend connection to IQSRVR.iqdb
   go
   resume connection to IQSRVR.iqdb
   go
   ```

Marking Tables to Prepare for Replication Testing

Mark tables in the primary database that you want to replicate to the Sybase IQ database

In these examples, `dbo` is the table owner of `testtab` in the `pdb1` primary database. `c1`, `c2`, and `c3` are columns in `testtab` with `int`, `int`, and `char(10)` datatypes, respectively.

1. Insert data rows into `testtab` for testing replication and verify the inserts are successful. For example, in Adaptive Server:

   ```
   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.
   - Adaptive Server – use the **sp_setrepdefmode** system procedure.
     - Adaptive Server 15.0.3 and later:

       ```
       sp_setrepdefmode testtab,'owner_on'
       go
       ```

     - Versions earlier than Adaptive Server 15.0.3:

       ```
       sp_setreptable testtab,'true', 'owner_on'
       go
       ```

   - Oracle – use the **pdb_setreptable** Replication Agent command:

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

     See *Replication Server Options > Replication Agent Administration Guide > Setup and Configuration > Primary Database Object Marking > Marking a Table in the Primary Database* for more usage information.

Creating Replication Definitions and Subscriptions

Create replication definitions and subscriptions for the tables marked for replication to Sybase IQ after you enable and configure RTL.

1. Create the *repdef_testtab* replication definition. Add any required referential constraint clauses to the replication definition to support RTL:
   - Adaptive Server:

     ```
     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
     ```

   - Oracle:

     ```
     create replication definition repdef_testtab
     with primary at ORA_DS.pdb1
     with primary table named 'TESTTAB'
     with replicate table named dbo.'testtab'
     (C1 as c1 int, C2 as c2 int, C3 as c3 char(10))
     primary key(C1)
     go
     ```

     Note: The default character case of Oracle is all upper case for object names. You can convert object names from upper to lower case in the replication definition, as shown in the example, or by using the **ltl_character_case** Replication Agent for Oracle configuration parameter. See *Replication Server Options > Replication Agent*

*Reference Manual > Configuration Parameters > Configuration Parameter Reference > **ltl_character_case**.*

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)
```

See also

- *Tables with Referential Constraints* on page 135

Verifying That RTL Works

Learn how to check that RTL works.

1. Log in to the primary data server and execute some operations, such as inserting new rows into `testtab`.

   For example, in Adaptive Server:

```
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
```

   You should see:

```
(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, you see:

```
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)
```

Migration from the Staging Solution to RTL

Migrate to the real-time loading solution if you are currently using the staging solution for replication to Sybase IQ.

The scenario assumes a replication topology where `pdb` is the primary database, PRS is the primary Replication Server, RRS is the replicate Replication Server, `staging_db` is the staging database, and `iqdb` is the replicate Sybase IQ database. The data flow in this scenario is:

```
pdb -----> PRS -----> RRS -----> staging_db -----> iqdb
```

Preparing to Migrate from the Staging Solution

Before you migrate from the staging solution, you need to perform some tasks.

1. You must upgrade both the primary and replicate Replication Servers to version 15.5 or later.

   See the *Replication Server Installation Guide* and *Replication Server Configuration Guide*.
2. Verify that no transactions flow into `pdb` 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 rssd_name
      ```

   c) Verify that the Replication Server queues have drained and that 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, you may lose data.
3. Verify that `pdb` 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 resynchronize the databases, you must purge and materialize `iqdb`.
4. Add an entry for the replicate Replication Server to the Sybase IQ `interfaces` file to allow the Sybase IQ server to connect to the replicate Replication Server and pull data.

Migrating to the Real-Time Loading Solution

Migrate from the staging solution to RTL.

1. Create a maintenance user in the 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 relevant connection profile and the maintenance user from step 1, such as *dbmaint*.
   - Adaptive Server:

     ```
     create connection to IQSRVR.iqdb
     using profile rs_ase_to_iq;standard
     set username to dbmaint
     set password to dbmaint
     go
     ```

   - Oracle:

     ```
     create connection to IQSRVR.iqdb
     using profile rs_oracle_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.
   - Adaptive Server
     - Adaptive Server 15.0.3 and later:

       ```
       sp_setrepdefmode testtab, 'owner_on'
       go
       ```

     - Versions earlier than Adaptive Server 15.0.3:

       ```
       sp_setreptable testtab, 'true', 'owner_on'
       go
       ```

   - Oracle

     ```
     pdb_setreptable testtab, mark, owner
     go
     ```

4. Recreate the replication definition to include owner information since you have enabled **owner_on** for Adaptive Server or **owner** for Oracle.
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.
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 directly from the primary data server to Sybase IQ.

See also

- *Tables with Referential Constraints* on page 135

Cleaning Up After Migration

Clean up the systems in the staging solution after enabling and configuring replication using RTL.

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.

Replication Server and Sybase IQ InfoPrimer Integration

Replication Server integrates with Sybase IQ InfoPrimer to support replication between a primary Adaptive Server database with a schema that is different from a replicate Sybase IQ database.

Sybase IQ InfoPrimer provides effective capabilities for transforming and loading data into a Sybase IQ database, but its extract capability lacks the real-time monitoring of Replication Server that is needed to maintain a replicate Sybase IQ database with the most current data. The Replication Server Real-Time Loading (RTL) feature uses bulk operation processing and compiled operations to achieve high-performance replication, but Replication Server lacks the data transformation and loading capabilities of Sybase IQ InfoPrimer. With the integration of Replication Server and Sybase IQ InfoPrimer, you can maintain a near real-time copy of Adaptive Server data in a replicate Sybase IQ database with different schema than the source. The integrated Replication Server and Sybase IQ InfoPrimer solution works in two parts: initial data materialization and ongoing data processing.

Materialization

The integrated Replication Server and Sybase IQ InfoPrimer solution performs a nonatomic bulk materialization of data from an Adaptive Server primary database to a replicate Sybase IQ database. The materialization is based on the Replication Server bulk materialization option and uses autocorrection where required.

Sybase IQ InfoPrimer creates staging tables on the replicate Sybase IQ database and performs the data-extract step of the materialization process on each primary database table.

Transformation stored procedures execute against the stage tables, and the result is written to base tables. The base tables, also known as end-user tables, are then used for business analysis.

Ongoing Data Processing

For specified tables, Replication Server uses the same staging tables and transformation stored procedures that were created in the materialization phase. Where possible, Replication Server compiles and loads operations to the staging tables, after which Replication Server executes the transformation stored procedures to update the base tables. In this way, Replication Server maintains a near real-time copy of data in the replicate Sybase IQ database.

Licensing

Special licensing requirements apply to the integration of Replication Server and Sybase IQ InfoPrimer. See *Replication Server New Features Guide > New Feature in Replication Server Version 15.6 ESD #1 > Licensing*.

Using the Replication Server and Sybase IQ InfoPrimer Integration

Use Sybase IQ InfoPrimer to materialize data into Sybase IQ with Replication Server materialization methods, and configure Replication Server to process updates made to primary data.

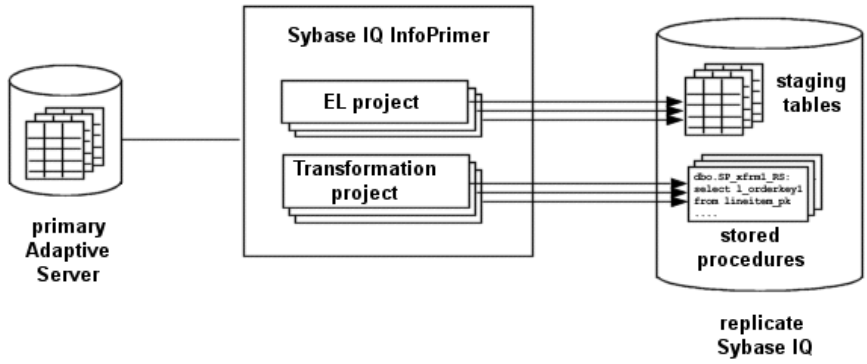
1. Before materialization:
   - Create an Extract and Load (EL) project in Sybase IQ InfoPrimer, selecting **Materialization with Replication Server**.

     In the RepServer tab of the EL project editor, you must also specify connection information for the primary Replication Server and the replicate Replication Server, if it is different from the primary. Sybase IQ InfoPrimer adds a command to the Processing tab. Do not modify or delete this command.

     For each source table, Sybase IQ InfoPrimer creates the required staging table definitions. Generate these staging tables on the replicate Sybase IQ database by selecting the **Create missing destination tables** icon on the Tables tab of the EL project editor.

     Note: If you are attempting to rematerialize, you must clear the `rs_status` table.
   - Create a SQL Transformation project, and model the transformation for each set of staging tables (insert, update, and delete) that have been generated in the replicate Sybase IQ database. Use the SQL Transformation project to deploy each set of transformations as a stored procedure in the replicate Sybase IQ database.

     Note: These transformation stored procedures truncate their corresponding staging tables when operations have been processed.

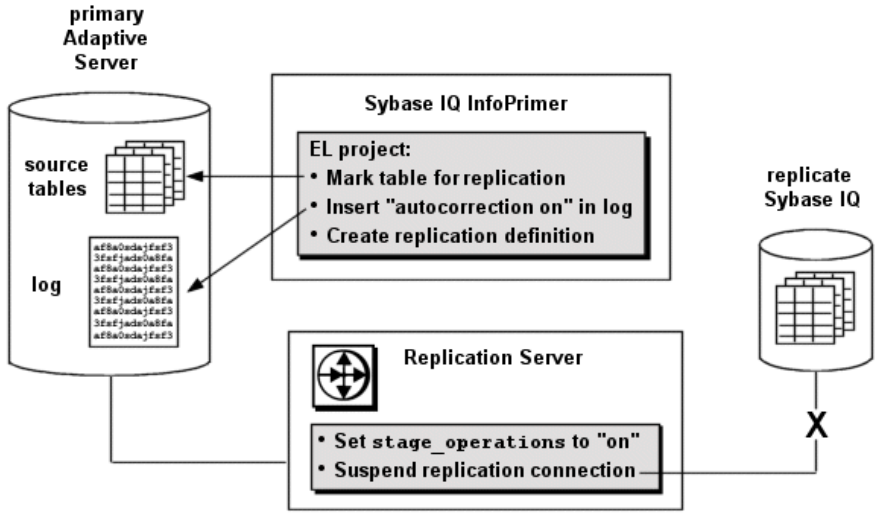


2. In your Replication Server instance, use the **stage_operations** connection parameter to configure the replicate database connection to stage operations for the tables specified in your EL project.

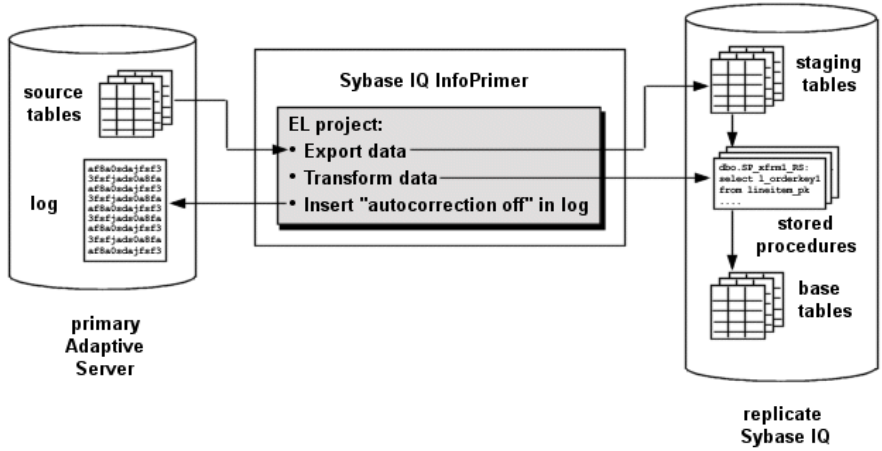
   Note: If **stage_operations** is set to on, Replication Server ignores the setting of **dsi_compile_enable** and enables RTL for the connection. Operations are compiled, as when **dsi_compile_enable** is set to on, and then staged.

   In Sybase IQ InfoPrimer, execute your EL project. For each primary table specified, the EL project:

   a) Marks the table for replication.
   b) Inserts an `autocorrection on` record in the primary database log, which results in suspension of the Replication Server replicate database connection.
   c) Creates a table replication definition in the RSSD.

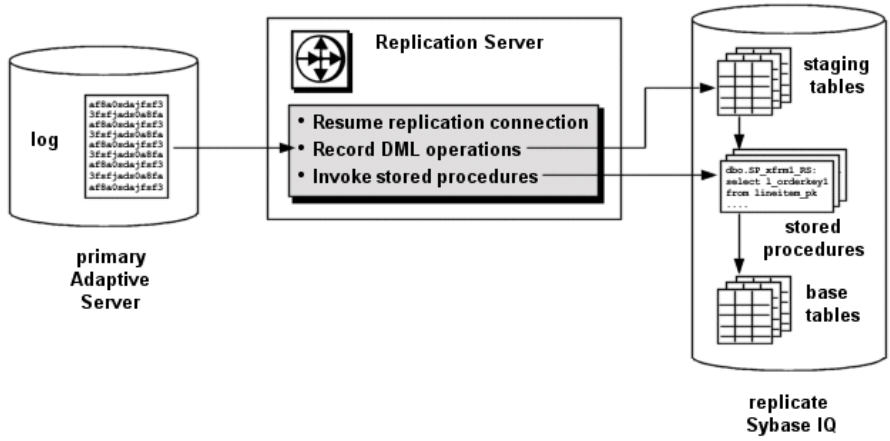


3. Your Sybase IQ InfoPrimer EL project exports primary data for each table into the corresponding staging tables on the replicate Sybase IQ, executes the transformation stored procedures, and inserts an `autocorrection off` record in the primary database log.



4. The Replication Server replicate database connection is resumed, and Replication Server processes any further changes to marked primary database tables using the staging tables and transformation stored procedures on the replicate Sybase IQ database.

Note: Sybase IQ InfoPrimer is only used for data migration and creating the staging tables and transformation stored procedures. It is not involved in replication.



Base Tables

Base tables contain data in its final form at the replicate Sybase IQ database.

Base table data can result from:

- SQL transformations – When the Replication Server replicate database connection has been configured to stage operations, the result of transformation stored procedures executing against the stage tables is written to the base tables.
- Replication – If a table has been excluded from staging, Replication Server bypasses the staging tables and replicates data directly to the base tables.

Staging Tables

If your Replication Server replicate database connection has been configured to stage operations logged for a primary table, these operations are compiled where possible and written to staging tables on the replicate Sybase IQ database.

For each table to be staged, there are three staging tables, each corresponding to DELETE, INSERT, and UPDATE operations:

- *owner_table_name*_DELETE_RS
- *owner_table_name*_INSERT_RS
- *owner_table_name*_UPDATE_RS

where *owner* and *table_name* are the owner and name of the corresponding primary database table. The names of these tables are generated by your EL project, and they cannot be changed.

Note: The Tables tab of your EL project displays only the insert staging table. However, the Table Creation window displays all three staging tables corresponding to a specified primary database table.

You must identify which primary database tables are to be staged in a Sybase IQ InfoPrimer EL project. You may also selectively exclude replicate tables from staging. For a table that has been excluded from staging, no corresponding staging tables need to be created, and data will be replicated from the primary table to a replicate table in the replicate Sybase IQ database.

If you configure a replicate database connection to stage tables but no staging tables exist in the replicate Sybase IQ database, the replicate database connection will be suspended. If a replication definition includes columns that are declared as identity columns, these will not be declared as identity columns in the corresponding staging tables.

Table Compilation

Compilation is not performed on noncompilable tables. Tables are considered noncompilable if they have RTL disabled, modified function strings, or minimal column replication enabled. Operations to noncompilable tables are captured in an ordered list and applied to the corresponding replicate table after compilation is complete.

Note: After Replication Server commits a staged operation, the transformation stored procedures truncate the corresponding staging tables. You should therefore not use the Replication Server **rs_subcmp** utility to validate staging tables.

Insert Staging Table Structure

Apart from changes and filtering applied by the corresponding replication definition, the insert staging table contains the same number of columns and the same column names as the primary table.

Delete Staging Table Structure

The delete staging table contains only the primary-key columns specified in the corresponding replication definition.

If no primary key is specified in the replication definition, the delete staging table contains all published columns except for:

- approximate numeric columns
- encrypted columns
- Java columns
- LOB columns

Note: Sybase recommends that you specify a primary key in your table replication definition to simplify processing and improve performance.

Update Staging Table Structure

The update staging table contains two columns for every primary-key column specified in the corresponding replication definition, one each for the column data before and after a change.

The update staging table also contains a column for each nonprimary-key column specified in the replication definition. To track whether changes have been made to data in these nonprimary-key columns, the update staging table contains one or more bitmap columns. Each bitmap column is of type `int` and can therefore track 32 non-primary key columns. A value of 1 constitutes a dirty bit, indicating that data has changed in the column corresponding to that bit position.

Note: The before-change and bitmap columns of the update staging table are not visible in the SQL Transformation project in Sybase IQ InfoPrimer.

Transformation Stored Procedures

For every primary database table that is staged, there should be a corresponding transformation stored procedure in the replicate Sybase IQ database. Replication Server executes these stored procedures against the staging tables, and the results are written to the base tables.

You must specify the transformations to be performed by these stored procedures in a Sybase IQ InfoPrimer SQL Transformation project and deploy the stored procedures to the replicate Sybase IQ database.

If you attempt to use stored procedures that do not exist in the replicate Sybase IQ database, or if a stored procedure fails to execute properly, the replicate database connection will be suspended.

Note: To ensure that you can see all the tables involved in a SQL Transformation project, do not select a schema in the project properties for the SQL Transformation project until you are ready to deploy your stored procedures to the replicate Sybase IQ database.

Parameters

Replication Server uses the **stage_operations** and **dsi_stage_all_ops** parameters to control table staging.

stage_operations

Set the **stage_operations** parameter of the **create connection** or **alter connection** command to have Replication Server write operations to staging tables for the specified connection.

You can configure staging for the replicate database connection. For example:

```
create connection to SYDNEY_IQ_RS.iq_db
 using profile rs_ase_to_iq;standard
 set username pubs2_maint
 set password pubs2_maint_pw
 set stage_operations to "on"
```

To selectively enable or disable staging for individual tables, use the **stage_operations** parameter of the **alter connection** command in reference to a specific replicate table. For example:

```
alter connection to SYDNEY_IQ_RS.iq_db
for replicate table named lineitem_5
set stage_operations to "off"
```

Here, Replication Server will not stage operations for the `lineitem_5` table but will instead replicate operations as normal.

Note: The **stage_operations** parameter can only be set for a connection to a Sybase IQ replicate (where the **dsi_dataserver_make** parameter is set to iq). The **dsi_dataserver_make** connection parameter is set appropriately when you use the Sybase IQ connection profile to create the connection.

dsi_compile_enable

If **stage_operations** is set to on, Replication Server ignores the setting of **dsi_compile_enable** and enables RTL for the connection. Operations are compiled, as when **dsi_compile_enable** is set to on, and then staged.

dsi_stage_all_ops

Use the **dsi_stage_all_ops** parameter of the **alter connection** command to prevent operation compilation for specified tables.

If table history must be preserved, as in the case of slowly changing dimension (SCD) tables, set **dsi_stage_all_ops** to on. For example:

```
alter connection to SYDNEY_IQ_RS.iq_db
for replicate table named lineitem_5
set dsi_stage_all_ops to "on"
```

Replication Server Components

Replication Server requires additional components to support the integration with Sybase IQ InfoPrimer.

The rs_status Table

The `rs_status` table stores information about the progress of materialization.

Column	Datatype	Description
`schema`	`varchar (255)`	Owner of table being materialized
`table-name`	`varchar (255)`	Name of table being materialized

Column	Datatype	Description
action	varchar (1)	• I – initial load • A – autocorrection phase • R – replication
start-time	time-stamp	Time action was started
endtime	time-stamp	Time action completed
status	varchar (1)	• P – action in progress • X – execution complete • E – execution error
pid	int	Reserved

For example, if autocorrection is in progress for `my_table`, `rs_status` contains a row like this:

```
schema tablename action starttime                endtime status pid
------ --------- ------ ----------------------- ------- ------ ---
sys    my_table  A      2011-07-11 19:11:25.531         P
```

If autocorrection is complete for `my_table`, `rs_status` contains a row like this:

```
schema tablename action starttime
------ --------- ------ -----------------------
sys    my_table  A      2011-07-11 19:11:25.531
```

```
endtime                 status pid
----------------------- ------ ---
2011-07-11 19:12:14.326 X
```

There is no automatic cleanup of `rs_status` data. Before you attempt to rematerialize a table, you must delete its corresponding row from `rs_status`:

```
delete rs_status where tablename=tablename and schema=owner
```

Autocorrection Functions

Replication Server uses the **rs_autoc_on**, **rs_autoc_off**, and **rs_autoc_ignore** functions to update the `rs_status` table.

See *Replication Server Reference Manual > Replication Server System Functions*.

System Variables

The **rs_autoc_on** and **rs_autoc_off** functions use two system variables when updating the `rs_status` table:

- *rs_deliver_as_name* – specifies the name of the replicate table affected by autocorrection.
- *rs_repl_objowner* – specifies the owner of the replicate table affected by autocorrection.

Default Datatype Translation

Sybase IQ supports all Adaptive Server datatypes in their native formats, so no Adaptive Server-to-Sybase IQ datatype translation is required.

Unsupported Features

The integration of Replication Server with Sybase IQ InfoPrimer is limited to certain features and platforms.

The integration of Replication Server with Sybase IQ InfoPrimer does not support:

- any replicate database other than Sybase IQ
- any primary database other than Adaptive Server
- replicated stored procedures
- custom function strings
- any pre-staging operation transformations other than those provided by RTL
- any transformations following those performed by the transformation stored procedures in the replicate Sybase IQ database

HANA DB as Replicate Data Server

Learn about the replicate database issues and considerations specific to the HANA DB data server in a Sybase replication system.

HANA DB Replicate Data Servers

You can replicate to a HANA DB data server using ExpressConnect for HANA DB.

ExpressConnect for HANA DB provides direct communication between Replication Server and HANA DB.

Replication Intrusions and Impacts on HANA DB

Replication Server creates objects in the replicate HANA DB instance to support Replication Server operations.

The created objects include:

- `rs_info`, which contains information about the sort order and character set used by the replicate database. 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 HANA DB data server. When a connection profile creates `rs_info`, the table entries are set to UTF-8 by default. To replicate unicode data, you must configure Replication Server to use UTF-8.

  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`, a table that contains information about replicated transactions applied to the replicate database. Each row in `rs_lastcommit` 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**. Replication Server creates a sequence named `rs_ticket_seq` by which Replication Server provides a unique row ID for `rs_ticket_history`. 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.

- `rs_status`, which stores information about the progress of direct load materialization. See the *Replication Server Reference Manual > Replication Server System Tables > rs_status*.
- `rs_ticket_seq`, which Replication Server uses to provide a unique row ID for `rs_ticket_history`.
- `rs_update_lastcommit`, which is a stored procedure used to update the `rs_lastcommit` table.
- `rs_version`, which is a Replication Server system table that stores version number information for the replication system.

Oracle RAW and LONG RAW as Primary Key Type

Do not use a column with the Oracle `RAW` or `LONG RAW` datatype as a primary key or as part of a primary key.

Unsupported Microsoft SQL Server Datatypes

These Microsoft SQL Server datatypes are not supported when the replicate database is HANA DB:

- varbinary(max)
- varchar(max)
- nvarchar(max)

Identity Columns

HANA DB does not have an `identity` type.

Make sure all `identity` columns in the primary map to the HANA DB `numeric` type. See *Datatype Translation and Mapping*.

HANA DB Replicate Database Permissions

Replication Server requires a maintenance user ID that you specify using the Replication Server **create connection** command to apply transactions in a replicate database.

The maintenance user ID must be defined at the HANA DB data server and granted authority to apply transactions in the replicate database. The maintenance user ID must have these schema privileges:

- **CREATE ANY** – allows the user to create tables, views, sequences, synonyms, SQL script functions, or database procedures in a schema.
- **DELETE**, **DROP**, **EXECUTE**, **INDEX**, **INSERT**, **SELECT**, and **UPDATE** – granted on every object stored in the specified schema.

ExpressConnect for HANA DB and Replicate Database Connectivity for HANA DB

Replication Server can connect to a HANA DB replicate database using ExpressConnect for HANA DB.

Using ExpressConnect for HANA DB

A Replication Server database connection to HANA DB can be:

- secure, in which the connection uses the **hdbuserstore** key specified in the database connection, or
- standard, in which the login requires a user ID, password, and host name. A standard connection requires an entry in the interfaces file.

Note: ExpressConnect for HANA DB supports Red Hat Enterprise Linux 6 and SuSE Linux Enterprise Server (SLES) 11 only.

The general format for an interfaces file entry for the HANA DB data server is:

```
[dataservername]
master tcp ether hostname port
query tcp ether hostname port
```

where *hostname* and *port* are the host and port number of the HANA DB dataserver, and *dataservername* is a label used to identify the host and port number. The interfaces file must be in standard format, not transport layer interface (TLI) format.

Each HANA DB installs with a unique instance number, and the port number is the instance number prefixed with 3 and suffixed with 15:

```
3in15
```

where *in* is the two-digit instance number. For example, the port number for a HANA DB with instance number 1 is 30115.

Configuring ExpressConnect for HANA DB

Configure ExpressConnect for HANA DB to set up connections between HANA DB and Replication Server.

Prerequisites

On UNIX and Linux systems, make sure the ODBC driver for HANA DB has write permission in the directory referenced by the $HOME environment variable.

ExpressConnect for HANA DB does not ship with the required HANA DB ODBC drivers. Before using ExpressConnect for HANA DB, download these libraries from the SAP Support MarketPlace, and install them after installing Replication Server. See the *Replication Server Release Bulletin > Special Installation Instructions > Installing ODBC Libraries for ExpressConnect for HANA DB*.

Task

1. Determine the HANA DB maintenance user ID and password used to connect from Replication Server. See *HANA DB Replicate Database Permissions* in the *Replication Server Heterogeneous Replication Guide*.

   Note: Before you create a connection to HANA DB, make sure that this maintenance user ID can log in to HANA DB and that the user ID has permission to create tables in HANA DB.

2. Use **isql** to create a connection to HANA DB from within Replication Server using the maintenance user ID and password combination. For example, for a standard connection and an Oracle primary database:

   ```
   create connection to
   <hana_server>.<hana_db>
   using profile rs_oracle_to_hana;ech
   set username <userid>
   set password <password>
   ```

   where:

   - *hana_server* is the case-sensitive name identifying the replicate HANA DB data server in your interfaces file (standard connection) or an **hdbuserstore** key (secure connection).
   - *hana_db* is an identifier for the HANA DB instance. This value is used by Replication Server only and is not used for connectivity.

     Note: HANA DB does not support multiple databases like Adaptive Server.

   For a secure connection and an Oracle primary database:

   ```
   create connection to
   <hana_server>.<hana_db>
   ```

```
using profile rs_oracle_to_hana;ech
set username <userid>
set password <password>
set dsi_connector_sec_mech to "hdbuserstore"
```

where **dsi_connector_sec_mech** specifies the DSI connector security mechanism.

See **create connection** in the *Replication Server Reference Manual*.

3. Update the `rs_info` table, if necessary:

```
update rs_info
set rsval = 'utf8'
where rskey = 'charset_name'
set rsval = 'bin_utf8'
where rskey = 'sortorder_name'
```

Licensing ExpressConnect for HANA DB

The license for ExpressConnect for HANA DB is distributed with Replication Server.

See the *Replication Server Installation Guide > Planning Your Installation > Obtaining a License at SPDC or SMP*.

Trace and Debug

Enable the tracing option in Replication Server to gather connector-level and connection-level diagnostic information.

Diagnostic information related to ExpressConnect for HANA DB execution is available for operations at both the connector level and the connection level, and for various diagnostic conditions. Not all conditions are available for both connector-level and connection-level tracing. Some also require the use of the diagnostic version of the ExpressConnect for HANA DB executable.

To enable tracing for ExpressConnect for HANA DB, see *ExpressConnect for Oracle Configuration Guide > Configuring ExpressConnect for Oracle > Trace and Debug*.

Character Set Conversion

ExpressConnect for HANA DB converts character sets depending the default values for the Replication Server **rs_charset**, **unicode_format**, and **dsi_charset_convert** parameters.

If **rs_charset** is set to iso_1, **unicode_format** is set to string, and **dsi_charset_convert** is set to on, ExpressConnect for HANA DB passes all data in its native form with no conversion. Unicode literals are not supported.

Set **rs_charset** to utf8 to replicate Unicode. Unicode literals are supported with this setting.

Note: Unicode replication, other than Unicode column types, requires **rs_charset** to be set to utf8.

If **rs_charset** is set to any supported character set, **unicode_format** is set to string (the default), and **dsi_charset_convert** is set to on, ExpressConnect for HANA DB converts all SQL language and character parameter data to UTF-16.

LOB Pointer Functions

ExpressConnect for Oracle and ExpressConnect for HANA DB do not use LOB pointers to manage LOB data. Consequently, the Replication Server system functions used to managing LOB pointers are unavailable to ExpressConnect for Oracle and ExpressConnect for HANA DB.

These functions–which include **rs_get_textptr**, **rs_textptr_init**, and **rs_writetext**–are visible to ExpressConnect, but their use is ignored by Replication Server.

HANA DB 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 HANA DB 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 HANA DB replicate database
  - Set connection properties

    You can connect using ExpressConnect for HANA DB, and the version or option name of the connection profile should be “ech.”
- Additional settings:
  - ExpressConnect 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 into HANA DB.

Function Strings, Error Classes, and User Defined Datatypes

Function strings are added to the Replication Server default `rs_hanadb_function_class`.

The function string class replaces several default Replication Server function strings with custom function strings designed to communicate with a HANA DB data server and access replication-related tables and procedures.

Warning! ExpressConnect for HANA DB supports the replication of LOB columns that use default Replication Server function strings for text and image processing. ExpressConnect for HANA DB does not support custom function strings for text and image processing. If you override the default Replication Server LOB processing with custom function strings for **rs_writetext**, **rs_textptr_init**, or other LOB-related function strings, text and image processing will fail.

Connection Profiles

Connection profiles allow you to configure your connection with a predefined set of properties.

Syntax

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

Parameters

data_server – Either:

- a reference to an interfaces file entry listing the host and port number of the HANA DB database, or
- the key name for the SAP Secure User Store entry for the HANA DB database

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 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. See the *Replication Server Reference Manual > Replication Server Commands >* ***create connection*** for a complete list of the other options for **create connection** command.

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**.

Class-Level Datatype Translations to RSSD

Class-level translations identify primary datatypes and the replicate datatypes the data must be translated into.

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

- `rs_ase_to_hanadb` – translates Adaptive Server datatypes to HANA DB datatypes.
- `rs_msss_to_hanadb` – translates Microsoft SQL Server datatypes to HANA DB datatypes.
- `rs_oracle_to_hanadb` – translates Oracle datatypes to HANA DB datatypes.
- `rs_udb_to_hanadb` – translates DB2 UDB datatypes to HANA DB datatypes.

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

```
create connection to hana_server.hana_db
using profile rs_ase_to_hana;ech
set username rs_maint_user
set password rs_maint_user_pwd
go
```

Objects in the HANA DB Replicate Database and Connection Properties

The connection profile creates the `rs_ticket_seq` sequence and the `rs_info`, `rs_lastcommit`, `rs_status`, and `rs_ticket_history` tables in the replicate database.

The connection profiles set these connection properties:

```
set error class rs_hanadb_error_class
set function string rs_hanadb_function_class
```

SAP Secure User Store

Use the SAP Secure User Store for non-ASE and non-IQ connectors, like ExpressConnect for HANA DB.

To support SAP Secure User Store logins, use the **dsi_connector_sec_mech** parameter with the **create connection** or **alter connection** commands. No interfaces file entry is required for the connection to the HANA DB instance when you use an SAP Secure User Store login.

The **dsi_connector_sec_mech** parameter is not a network-based security parameter. It is valid only for connections.

To use **dsi_connector_sec_mech** with ExpressConnect for HANA DB, you must first use the **hdbuserstore** utility to create a secure user store of encrypted credentials. For example:

```
hdbuserstore set hanads myhost:30215 myuser mYpA5Sw0rD
```

where "hanads" is a label used as the key for querying the secure user store, "myhost:30215" is the connection environment host name and port number, "myuser" is the user ID, and "mYpA5Sw0rD" is the password.

Note: The secure store must be created with the same operating system user ID that starts and runs Replication Server. Otherwise, Replication Server cannot access the secure user store.

Once you have created a secure user store, you can create a connection to the HANA DB instance with the encrypted credentials. For example, to connect to the HANA DB instance for ASE-to-HANA DB replication:

```
create connection to hanads.hanadb
using profile rs_ase_to_hanadb;ech
set username "foo"
set password "bar"
set dsi_connector_sec_mech to "hdbuserstore"
go
```

where the user ID and password "foo" and "bar" are unused values supplied only to satisfy the syntax of the **create connection** command.

Note: The same operating system user who started Replication Server must also own the secure user store.

You can alter an existing connection to a HANA DB instance to use **dsi_connector_sec_mech**. For example:

```
alter connection to hanads.hanadb
set dsi_connector_sec_mech to "hdbuserstore"
go
```

After running the **alter connection** command, you must suspend and resume the connection:

```
suspend connection to hanads.hanadb
go
resume connection to hanads.hanadb
go
```

Additional Settings

Learn about the additional settings provided to support replication.

The settings include:

- Command Batching settings
- Dynamic SQL settings
- HVAR settings

Command Batching

HANA DB does not support command batching. Do not turn on command batching for the HANA DB database connection.

Dynamic SQL

The **dynamic_sql** configuration parameter is set to 'on' by default, and this setting is recommended for all HANA DB connection profiles.

HVAR

If you have a Replication Server license for Advanced Service Options, you can use Replication Server High-Volume Adaptive Replication (HVAR) in replicating to HANA DB. See the *Replication Server Administration Guide Volume 2 > Performance Tuning > Advanced Services Option > High Volume Adaptive Replication to Adaptive Server*.

Heterogeneous Multi-Path Replication

Use multiple replication paths to increase replication throughput and performance, and reduce contention.

In a single-path replication environment, transactions replicate serially from the primary database to the replicate database to ensure the primary database transaction commit order, and therefore to ensure that the replicate database is consistent with the primary database. The serial mode of applying transactions to the replicate database remains, even though multiple applications typically execute their respective transactions in parallel at the primary database, or even if there are transactions arriving from multiple primary databases.

There are replication environments that can maintain data consistency within a subset of tables, without serializing all transactions that originate from the same primary database. A typical example of this environment is when different applications that access different sets of data modify a single primary database. The different sets of data within the subset of tables that are modified by a specific application continue to replicate serially. Data in different subsets of tables can replicate in parallel.

Multi-Path Replication™ supports the replication of data through different streams, while still maintaining data consistency within a path, but not adhering to the commit order across different paths.

A replication path encompasses all the components and modules between the Replication Server and the primary or replicate database. In multipath replication, you can create multiple primary replication paths for multiple Replication Agent connections from a primary database to one or more Replication Servers, and multiple replicate paths from one or more Replication Servers to the replicate database. You can configure multi-path replication in warm standby and multisite availability (MSA) environments. You can convey transactions over dedicated routes between Replication Servers to avoid congestion on shared routes, and you can dedicate an end-to-end replication path from the primary database through Replication Servers to the replicate database, to objects such as tables and stored procedures.

See *Replication Server > Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication* to configure Adaptive Server as the primary or replicate database in a multipath replication system.

License

Multi-Path Replication is licensed as part of the Advanced Services Option. See *Replication Server Installation Guide > Planning Your Installation > Obtaining a License*.

System Requirements

Table 1. Primary and Replicate Database Pairs Supported in Heterogeneous Multi-Path Replication Systems

Primary Database	Replicate Database
Adaptive Server	Adaptive Server
Adaptive Server	HANA DB
Adaptive Server	Oracle
Adaptive Server	Sybase IQ
Oracle	Adaptive Server
Oracle	HANA DB
Oracle	Oracle
Oracle	Sybase IQ

Table 2. Multi-Path Replication Database Versions Supported

Database	Supported Versions
Adaptive Server	15.7 and later
HANA DB	1.0 SP5
Oracle	Oracle 10g and 11g. See *Replication Server Options > Replication Agent Release Bulletin > Product Summary > Product Compatibility*.
Sybase IQ	15.1 and later. See *Replication Server Release Bulletin > Product Compatibility > Replication Server Interoperability* .

Parallel Transaction Streams

Multi-path replication can improve replication performance as long as transactions can be divided into parallel streams, and do not have to be serially committed across different streams.

You can improve replication performance by dividing transactions into parallel replication paths to reduce congestion. You can divide transactions according to parallelization rules such as transaction attributes or derived data values. For example, you can:

- Dedicate paths to specific objects such as tables or stored procedures. When you bind an object to a path, Replication Agent sends any replicable actions that you perform upon that object through the path to the Replication Servers that you define in your multiple replication path configuration. RepAgent for Adaptive Server and Replication Agent for Oracle support this replication distribution mode.
- Divide transactions by the session ID of the client connection on the primary database. RepAgent for Adaptive Server supports distributing transactions by client connections.
- Dedicate a Replication Server to each path.
- Allocate dedicated paths or less congested paths for priority replication by binding objects to these paths or by creating dedicated routes between Replication Servers.

Default and Alternate Connections

In multipath replication, connections include the default and one or more alternate connections.

A connection that accepts data from a Replication Agent is a primary connection, and a connection that applies data to a database is replicate connection. A default or alternate connection can be a primary or replicate connection.

The default connection is the one that you create from a Replication Server to a specific primary or replicate database when you add the database to the replication domain. You can use **rs_init**, the Replication Manager Sybase Central plug-in, **create connection**, or **create connection ... using profile** to create the default connection depending on whether the data server is Adaptive Server or a supported non-ASE data server.

The default connection uses the data server and database names in the form of *dataserver.database* as the connection name, where *dataserver* and *database* are the actual data server and database names, respectively.

You can create multiple alternate connections after you create the default connection, which is required. Each alternate connection must have a unique name.

After you create an alternate connection, you can alter the connection properties or drop the connection. You can also display the status of all connections and create subscriptions for the connection.

When you create an alternate connection, the user ID must be a valid user. When you create connections to Sybase IQ replicate databases, you must specify the connection profile and connection profile version, and a unique maintenance user name for the default connection and each alternate connection .

Interfaces File Requirements for Sybase IQ

Create an entry for the Replication Server in the `interfaces` file (`sql.ini` file in Windows) of the Sybase IQ data server. If there are multiple Replication Servers replicating to a single Sybase IQ server, you must add all Replication Servers to the `interfaces` file.

If you are creating connections to different Sybase IQ multiplex nodes, create entries for each of the affected nodes in the `interfaces` file of the replicate Replication Server.

Dedicated Routes

A dedicated route distributes only transactions for a specific primary connection. You can create a dedicated route to the replicate Replication Server to replicate high priority transactions or to maintain a less congested path for a specific primary connection.

A shared route is between a primary Replication Server and a replicate Replication Server that distributes transactions for all the primary connections originating from the primary Replication Server. You do not bind shared routes to a specific connection. Connections that you do not bind to a dedicated route use any available valid shared route.

You can create a dedicated route only if:

- A shared route exists from the primary Replication Server to the destination Replication Server and the shared route is a direct route. You cannot create a dedicated route if there is only an indirect route between the Replication Servers.
- The shared route is valid and not suspended.
- The route version of the shared route is 1570 or later.

Creating Dedicated Routes

Use **create route** and the **with primary at** clause to create a dedicated route.

For example, to create a dedicated route between the RS_NY primary Replication Server and the RS_LON replicate Replication Server for the `NY_DS.pdb1` primary connection, at RS_NY enter:

```
create route to RS_LON
 with primary at NY_DS.pdb1
 go
```

After you create a dedicated route for a specific connection, all transactions from the connection to the destination Replication Server follow the dedicated route.

Commands to Manage Dedicated Routes

Use **create route**, **drop route**, **resume route**, and **suspend route** to manage and monitor dedicated routes.

Include the **with primary at** *dataserver.database* clause in the command to specify a dedicated route, where *dataserver.database* is the primary connection name.

See **create route**, **drop route**, **suspend route**, and **resume route** in *Replication Server Reference Manual > Replication Server Commands*.

Command	Syntax	Command and Parameter Changes
create route	`create route to dest_replication_server { with primary at dataserver.database | set next site [to] thru_replication_server | [set username [to] user] [set password [to] passwd] [set route_param to 'value' [set route_param to 'value']...] [set security_param to 'value' [set security_param to 'value']...]}`	If you specify a user ID when you create a dedicated route, the user ID must be a valid user.

Command	Syntax	Command and Parameter Changes
drop route	`drop route to dest_replication_server [with primary at dataserver.database] [with nowait]`	You must drop the dedicated route before you drop a shared route. After you drop a dedicated route, transactions from the specified primary connection to the destination Replication Server go through the shared route. **Warning!** Use the **with nowait** clause only as a last resort. The clause forces Replication Server to drop a route even if the route contains transactions in the outbound queue of the route. As a result, Replication Server may discard some transactions from the primary connections. The clause instructs Replication Server to drop the dedicated route even if the route cannot communicate with the destination Replication Server. If you use the clause, use **sysadmin purge_route_at_replicate** at the former destination site to remove subscriptions and route information from the system tables at the destination. See *Replication Server Administration Guide Volume 1 > Manage Routes > Drop Routes > **drop route** comand*.
suspend route	`suspend route to dest_replication_server [with primary at dataserver.database]`	
resume route	`resume route to dest_replication_server [with primary at dataserver.database] [skip transaction with large message]`	

Display Dedicated Route Information

Use **admin who** to display information on dedicated routes between Replication Servers.

In this example, there is a dedicated route from the RS_NY primary Replication Server to the RS_LON replicate Replication Server for the `NY_DS.pdb1` primary connection. Enter **admin who** at the two Replication Servers and you see:

- At `RS_LON`:

```
Spid Name       State               Info
45 SQT          Awaiting Wakeup     103:1  DIST NY_DS.pdb1
13 SQM          Awaiting Message    103:1 NY_DS.pdb1
32 REP AGENT    Awaiting Command    NY_DS.pdb1
16 RSI          Awaiting Wakeup     RS_LON
11 SQM          Awaiting Message    16777318:0 RS_LON
55 RSI          Awaiting Wakeup     RS_LON(103) /* Dedicated RSI
thread */
53 SQM          Awaiting Message    16777318:103 RS_LON(103) /
*Dedicated RSI outbound queue */
```

- At `RS_NY`:

```
Spid Name      State                Info
37 RSI USER   Awaiting Command    RS_NY(103) /*Dedicated RSI user
*/
32 RSI USER    Awaiting Command    RS_NY
```

See *Replication Server Reference Manual > Replication Server Commands >* ***admin who***.

Heterogeneous Multi-Path Replication Scenarios

Use the scenarios to build multiple replication paths between Adaptive Server, Oracle, and Sybase IQ databases.

Multi-Path Replication from Adaptive Server to HANA DB

Set up a Multi-Path Replication system with two primary and replicate paths for end-to-end replication from an Adaptive Server primary to a HANA DB replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Adaptive Server database and RepAgent thread
- Replication Server
- ExpressConnect for HANA DB
- a replicate HANA DB database

Task

1. At the primary Adaptive Server database, select or create two sets of tables or stored procedures that you want to replicate through two replication paths. These transaction sets must be divisible into parallel replication paths.
2. Create a default connection to the primary Adaptive Server database.

   Use **create connection** or **rs_init** to create the default connection. See *Replication Server Configuration Guide > Configure Replication Server and Add Databases using* **rs_init**.
3. Enable Multithreaded Adaptive Server RepAgent and Multiple Paths for Adaptive Server RepAgent.

   You can also set the memory and send buffers available to Adaptive Server RepAgent. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Enabling Multithreaded RepAgent and Multiple Paths for RepAgent*.

   a) Enable multithreaded RepAgent.
   At the primary Adaptive Server, enter:

   ```
   sp_config_rep_agent pdb, 'multithread rep agent', 'true'
   ```

   where *pdb* is the primary Adaptive Server database.

   b) Set the number of replication paths for RepAgent.
   For example, to enable two paths, enter:

   ```
   sp_config_rep_agent pdb, 'max number of replication paths', '2'
   ```
4. Create an alternate replication path from the primary database.

   See *Replication Server > Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Multiple Primary Replication Paths*.

   a) Create an alternate replication path named `pdb_conn2` between the primary database and Replication Server.
   At the primary database, enter:

   ```
   sp_replication_path "pdb", 'add',
   "pdb_conn2", "PRS",
   "muser", "mpwd"
   ```

   where:

   - *PRS* is the Replication Server.
   - *muser* is the maintenance user.
   - *mpwd* is the maintenance user password.

   (Optionally) Create logical paths that you can use to distribute data and objects bound to a physical path to multiple Replication Servers. See *Replication Server Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Logical Primary Replication Paths* .

b) Create the corresponding alternate primary connection in Replication Server, and use the same Adaptive Server data server name with the alternate path name—`pdb_conn2`.
At the Replication Server, enter:

```
create alternate connection to pds.pdb
named pds.pdb_conn2
set error class to rs_sqlserver_error_class
set function string class to rs_sqlserver_function_class
set username to muser
set password to mpwd
with primary only
```

where *pds* is the primary Adaptive Server.

The replication system contains two primary replication paths—the default and `pdb_conn2`.

5. Use **admin show_connections, 'primary'** to display the primary connections you have created.
6. At the replicate HANA DB instance, create a maintenance user for the replicate HANA DB instance:

```
CREATE USER muser PASSWORD mpwd;
```

7. Grant these authorities to the maintenance user:
   - **CREATE ANY**
   - **DELETE**
   - **DROP**
   - **EXECUTE** (on all replicate stored procedures)
   - **INDEX**
   - **SELECT**
   - **UPDATE** (on all replicate tables)
8. If you are connecting to HANA DB with a standard connection and not using SAP Secure User Store, add an entry to your interfaces file identifying the replicate HANA DB data server, and restart Replication Server:

```
[dataservername]
master tcp ether hostname port
query tcp ether hostname port
```

where *hostname* and *port* are the host and port number of the HANA DB dataserver, and *dataservername* is a label used to identify the host and port number.

If you are using SAP Secure User Store, create a user store of encrypted credentials:

```
hdbuserstore set rds myhost:xxxxx my_securestore_user
my_securestore_pwd
```

where

- *rds* is the key for the secure store entry
- *myhost*:*xxxxx* is the connection environment host name and port number

- *my_securestore_user* and *my_securestore_pwd* are SAP Secure User Store credentials

9. Create a connection to the replicate HANA DB instance using ExpressConnect for HANA DB.

   For a standard connection:

   ```
   create connection to rds.rdb
   using profile rs_ase_to_hanadb;ech
   set username muser
   set password mpwd
   go
   ```

   For SAP Secure User Store:

   ```
   create connection to rds.rdb
   using profile rs_ase_to_hanadb;ech
   set username auser
   set password apwd
   set dsi_connector_sec_mech to "hdbuserstore"
   go
   ```

   where:

   - *rds* is the replicate HANA DB data server. For a standard connection, this must match the interfaces file entry. For an SAP Secure User Store connection, this must match what you used as the key to create a user store of encrypted credentials with the **hdbuserstore** utility.
   - *rdb* is placeholder: You must provide a value, but it is not used..
   - *muser* is the maintenance user for the replicate HANA DB instance.
   - *mpwd* is the replicate HANA DB maintenance user password.
   - *auser* and *apwd* are unused values supplied only to satisfy the syntax of the **create connection** command.

   Note: The secure store must be created with the same operating system user ID that starts and runs Replication Server. Otherwise, Replication Server cannot access the secure user store.

10. Create an alternate connection to the replicate HANA DB database through ExpressConnect for HANA DB.

    For a standard connection:

    ```
    create alternate connection to rds.conn2
    using profile rs_ase_to_hanadb;ech
    set username muser
    set password mpwd
    go
    ```

    For SAP Secure User Store:

    ```
    create connection to rds.conn2
    using profile rs_ase_to_hanadb;ech
    set username auser
    set password apwd
    ```

```
set dsi_connector_sec_mech to "hdbuserstore"
go
```

11. At the primary database, mark the `ptab1` and `ptab2` tables for replication.

```
sp_setreptable ptab1,'true'
go
sp_setreptable ptab2,'true'
go
```

12. Set the distribution mode at the primary database to distribute by object binding:

```
sp_config_rep_agent pdb, 'multipath distribution model', 'object'
```

13. Quiesce Replication Server and restart RepAgent.

    See *Replication Server Administration Guide Volume 1 > Manage a Replication System > Quiesce Replication Server > Quiescing a Replication System*.

14. Bind the `ptab2` table to the `pdb_conn2` alternate connection.

```
sp_replication_path pdb, 'bind', "table", "dbo.ptab2",
"pdb_conn2"
```

    You can only bind objects to alternate replication paths. All objects that you do not bind to an alternate replication path, such as the `ptab1` table, use the default path instead.

15. Create two replication definitions against the primary Adaptive Server database:
    For example to create the `ptab1_repdef` replication definition for the `ptab1` table:

```
create replication definition ptab1_repdef
with primary at pds.pdb
with all tables named 'ptab1'
...
go
```

    To create the `ptab2_repdef` replication definition for the `ptab2` table:

```
create replication definition ptab2_repdef
with primary at pds.pdb
with all tables named 'ptab2'
...
go
```

    Note: These replication definitions must use the default primary connection name.

    If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

16. Create a subscription against the default replicate connection.
    For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

```
create subscription ptab1_sub
for ptab1_repdef
with replicate at rds.rdb
without materialization
go
```

17. Create a subscription against the alternate replicate connection.

For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

```
create subscription ptab2_sub
for ptab2_repdef
with replicate at rds.conn2
without materialization
go
```

Multi-Path Replication from Adaptive Server to Oracle

Set up a Multi-Path Replication system with two primary and replicate paths for end-to-end replication from an Adaptive Server primary to an Oracle replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Adaptive Server database and RepAgent thread
- Replication Server
- ExpressConnect for Oracle
- a replicate Oracle database

Task

1. At the primary Adaptive Server database, select or create two sets of tables or stored procedures that you want to replicate through two replication paths. These transaction sets must be divisible into parallel replication paths.
2. Create a default connection to the primary Adaptive Server database.

   Use **create connection** or **rs_init** to create the default connection. See *Replication Server Configuration Guide > Configure Replication Server and Add Databases using* **rs_init**.
3. Enable Multithreaded Adaptive Server RepAgent and Multiple Paths for Adaptive Server RepAgent.

   You can also set the memory and send buffers available to Adaptive Server RepAgent. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Enabling Multithreaded RepAgent and Multiple Paths for RepAgent*.

   a) Enable multithreaded RepAgent.
   At the primary Adaptive Server, enter:

   ```
   sp_config_rep_agent pdb, 'multithread rep agent', 'true'
   ```

   where *pdb* is the primary Adaptive Server database.

   b) Set the number of replication paths for RepAgent.
   For example, to enable two paths, enter:

   ```
   sp_config_rep_agent pdb, 'max number of replication paths', '2'
   ```

4. Create an alternate replication path from the primary database.

   See *Replication Server > Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Multiple Primary Replication Paths*.

   a) Create an alternate replication path named `pdb_conn2` between the primary database and Replication Server.
   At the primary database, enter:

   ```
   sp_replication_path "pdb", 'add',
   "pdb_conn2", "PRS",
   "muser", "mpwd"
   ```

   where:
   - *PRS* is the Replication Server.
   - *muser* is the maintenance user.
   - *mpwd* is the maintenance user password.

   (Optionally) Create logical paths that you can use to distribute data and objects bound to a physical path to multiple Replication Servers. See *Replication Server Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Logical Primary Replication Paths* .

   b) Create the corresponding alternate primary connection in Replication Server, and use the same Adaptive Server data server name with the alternate path name—`pdb_conn2`.
   At the Replication Server, enter:

   ```
   create alternate connection to pds.pdb
   named pds.pdb_conn2
   set error class to rs_sqlserver_error_class
   set function string class to rs_sqlserver_function_class
   set username to muser
   set password to mpwd
   with primary only
   ```

   where *pds* is the primary Adaptive Server.

   The replication system contains two primary replication paths—the default and `pdb_conn2`.

5. Use **admin show_connections, 'primary'** to display the primary connections you have created.
6. Copy the `tnsnames.ora` file for your replicate Oracle database to the Replication Server `RS_installation_directory\ REP-15_5\connector\oraoci \network\admin` directory, and restart Replication Server.
7. Create a connection to the replicate Oracle database through ExpressConnect for Oracle.

   ```
   create connection to tns_alias_name.rdb
   using profile rs_oracle_to_oracle;eco
   set username muser
   set password mpwd
   set dsi_dataserver_make to 'ora'
   ```

```
set dsi_connector_type to 'oci'
set batch to 'off'
go
```

where:

- *tns_alias_name* is the alias name for the replicate Oracle database defined in the `tnsnames.ora` file for the replicate Oracle database.
- *rdb* is the replicate Oracle System ID (SID) that is paired with the above **tns_alias_name** in the `tnsnames.ora` file. The default value is ORCL.
- *muser* is the maintenance user for the replicate Oracle database.
- *mpwd* is the replicate Oracle maintenance user password.

See the *ExpressConnect for Oracle Installation and Configuration Guide* for more information about naming the default connection.

8. Create an alternate connection to the replicate Oracle database through ExpressConnect for Oracle.

```
create alternate connection to tns_alias_name.rdb
named tns_alias_name.conn2
set error class rs_oracle_error_class
set function string class rs_oracle_function_class
set username muser
set password mpwd
set dsi_dataserver_make to 'ora'
set dsi_dataserver_type to 'oci'
set batch to 'off'
set dsi_proc_as_rpc to 'on'
go
```

9. At the primary database, mark the `ptab1` and `ptab2` tables for replication.

```
sp_setreptable ptab1,'true'
go
sp_setreptable ptab2,'true'
go
```

10. Set the distribution mode at the primary database to distribute by object binding:

```
sp_config_rep_agent pdb, 'multipath distribution model', 'object'
```

11. Quiesce Replication Server and restart RepAgent.

    See *Replication Server Administration Guide Volume 1 > Manage a Replication System > Quiesce Replication Server > Quiescing a Replication System*.

12. Bind the `ptab2` table to the `pdb_conn2` alternate connection.

```
sp_replication_path pdb, 'bind', "table", "dbo.ptab2",
"pdb_conn2"
```

    You can only bind objects to alternate replication paths. All objects that you do not bind to an alternate replication path, such as the `ptab1` table, use the default path instead.

13. Create two replication definitions against the primary Adaptive Server database:
    For example to create the `ptab1_repdef` replication definition for the `ptab1` table:

```
create replication definition ptab1_repdef
with primary at pds.pdb
```

```
with all tables named 'ptab1'
...
go
```

To create the `ptab2_repdef` replication definition for the `ptab2` table:

```
create replication definition ptab2_repdef
with primary at pds.pdb
with all tables named 'ptab2'
...
go
```

Note: These replication definitions must use the default primary connection name.

If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

14. Create a subscription against the default replicate connection.
    For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

```
create subscription ptab1_sub
for ptab1_repdef
with replicate at tns_alias_name.rdb
without materialization
go
```

15. Create a subscription against the alternate replicate connection.
    For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

```
create subscription ptab2_sub
for ptab2_repdef
with replicate at tns_alias_name.conn2
without materialization
go
```

See also

- *Oracle as Replicate Data Server* on page 103

Multi-Path Replication from Adaptive Server to Sybase IQ

Set up a multipath replication replication system with two primary and replicate connections for end-to-end replication from an Adaptive Server primary to a Sybase IQ replicate database.

The replication system in this scenario consists of the `pdb` database in the ASE_DS primary Adaptive Server, the IQSRVR replicate Sybase IQ data server containing the `iqdb` database, the PRS primary Replication Server, and the `testtab1` and `testtab2` tables.

1. At the `pdb` primary database, select or create two sets of tables or stored procedures that you want to replicate through two replication paths.
2. Create the default connection to the `pdb` primary Adaptive Server database.

Use **create connection** or **rs_init** to create the default connection. See *Replication Server Configuration Guide > Configure Replication Server and Add Databases using* ***rs_init***.

3. Enable Multithreaded Adaptive Server RepAgent and Multiple Paths for Adaptive Server RepAgent.

   You can also set the memory and send buffers available to Adaptive Server RepAgent. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Enabling Multithreaded RepAgent and Multiple Paths for RepAgent*.

   a) Enable multithreaded RepAgent.
   At the primary Adaptive Server, enter:

   ```
   sp_config_rep_agent pdb, 'multithread rep agent', 'true'
   ```

   b) Set the number of replication paths for RepAgent.
   For example, to enable two paths, enter:

   ```
   sp_config_rep_agent pdb, 'max number of replication paths', '2'
   ```

4. Create an alternate replication path from the `pdb` primary database to the PRS Replication Server.

   See *Replication Server > Administration Guide: Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Multiple Primary Replication Paths*.

   a) Create an alternate replication path for the Adaptive Server RepAgent named `pdb_conn2` between `pdb` and PRS Replication Server.
   At the primary Adaptive Server, enter:

   ```
   sp_replication_path "pdb", 'add',
   "pdb_conn2", "PRS",
   "dbmaint2", "dbmaint2pwd"
   ```

   (Optionally) Create logical paths that you can use to distribute data and objects bound to a physical path to multiple Replication Servers. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Multiple Primary Replication Paths > Creating Logical Primary Replication Paths* .

   b) Create the corresponding alternate primary connection in Replication Server and use the same Adaptive Server data server name with the alternate path name—`pdb_conn2`:
   At the Replication Server, enter:

   ```
   create alternate connection to ASE_DS.pdb
   named ASE_DS.pdb_conn2
   set error class to rs_sqlserver_error_class
   set function string class to rs_sqlserver_function_class
   set username to dbmaint2
   set password to dbmaint2pwd
   with primary only
   ```

   The replication system contains two primary replication paths: the default—`ASE_DS.pdb`, and `ASE_DS.pdb_conn2`.

5. Use **admin show_connections, 'primary'** to display the primary connections you have created.
6. Create the default replicate connection to the Sybase IQ database with dbmaint1 as the maintenance user.

   You must specify the connection profile and connection profile version, and a unique maintenance user name for the default connection and each alternate connection.

   ```
   create connection to IQSRVR.iqdb
   using profile rs_ase_to_iq;standard
   set username to dbmaint1
   set password to dbmaint1
   go
   ```
7. Verify that the connection is running with **admin who**:
8. Create an alternate replicate connection named `IQSRVR.pdb_conn2` to the `iqdb` Sybase IQ database with dbmaint2 as the maintenance user.

   ```
   create alternate connection to IQSRVR.iqdb
   named IQSRVR.pdb_conn2
   using profile rs_ase_to_iq;standard
   set username to dbmaint2
   set password to dbmaint2pwd
   go
   ```

   (Optionally) Create alternate connections to available Sybase IQ multiplex nodes. Ensure that the connection names are unique.

   For example, to create the `pdb_conn3` alternate connection to the `iqdb2` database in the IQSRVR2 Sybase IQ node, enter:

   ```
   create alternate connection to IQSRVR2.pdb_conn3
   named IQSRVR2.iqdb2_conn1
   using profile rs_ase_to_iq;standard
   set username to dbmaint3
   set password to dbmaint3pwd
   go
   ```
9. Display the replicate connections you have created with **admin show_connections, 'replicate'**.
10. Enable RTL for the default and alternate replicate connections to Sybase IQ.
    a) Enable RTL for the default connection:

       ```
       alter connection to IQSRVR.iqdb
       set dsi_compile_enable to 'on'
       go
       ```
    b) Enable RTL for the `IQSRVR.pdb_conn2` alternate connection:

       ```
       alter connection to IQSRVR.pdb_conn2
       set dsi_compile_enable to 'on'
       go
       ```
    c) Suspend and resume the connections to the replicate Sybase IQ database:

       ```
       suspend connection to IQSRVR.iqdb
       go
       suspend connection to IQSRVR.pdb_conn2
       ```

```
go
resume connection to IQSRVR.iqdb
go
resume connection to IQSRVR.pdb_conn2
go
```

Restart Replication Server if you use **configure replication server** to enable RTL or if you want to suspend and resume all connections at the same time.

11. Optionally set values for parameters to configure RTL processing and tune RTL performance.
12. At the primary database, mark the `testtab1` and `testtab2` tables for replication.

```
sp_setreptable testtab1,'true'
go
sp_setreptable testtab2,'true'
go
```

13. Set the distribution mode at the primary database to distribute by object binding:

```
sp_config_rep_agent pdb, 'multipath distribution model', 'object'
```

14. Quiesce Replication Server and restart RepAgent.

    See *Replication Server Administration Guide Volume 1 > Manage a Replication System > Quiesce Replication Server > Quiescing a Replication System*.

15. Bind the `testtab2` table to the `ASE_DS.pdb_conn2` alternate connection.

```
sp_replication_path pdb, 'bind', "table", "dbo.testtab2",
"pdb_conn2"
```

    You can only bind objects to alternate replication paths. All objects that you do not bind to an alternate replication path, such as the `testtab1` table, use the default path instead.

16. Create replication definitions for the marked tables. Add any required referential constraint clauses to the replication definition to support RTL:
    To create the **repdef_testtab1** replication definition for the `testtab1` table, enter:

```
create replication definition repdef_testtab1
with primary at ASE_DS.pdb1
with primary table named 'testtab1'
with replicate table named dbo.'testtab1'
(c1 int, c2 int, c3 char(10))
primary key(c1)
go
```

    To create the **repdef_testtab2** replication definition for the `testtab2` table, enter:

```
create replication definition repdef_testtab2
with primary at ASE_DS.pdb1
with primary table named 'testtab2'
with replicate table named dbo.'testtab2'
(c1 int, c2 int, c3 char(10))
primary key(c1)
go
```

    All replication definitions refer to the default primary connection.

17. Create a subscription to match each table replication definition and specify the connection:
    a) Create the **sub_testtab1** subscription for the **repdef_testtab1** replication definition and specify the default connection to replicate transactions:

```
create subscription sub_testtab1 for repdef_testtab1
with replicate at IQSRVR.iqdb
without materialization
go
```

    b) Create the **sub_testtab2** subscription for the **repdef_testtab2** replication definition and specify the `IQSRVR.iqdb_conn2` alternate connection to replicate transactions:

```
create subscription sub_testtab2 for repdef_testtab2
with replicate at IQSRVR.pdb_conn2
without materialization
go
```

    See *Replication Server > Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Replication Definitions and Subscriptions > Moving Subscriptions Between Connections*.

See also

- *Sybase IQ as Replicate Data Server* on page 113
- *RTL Performance Tuning* on page 125

Multi-Path Replication from Oracle to Adaptive Server

Set up a Multi-Path Replication system with two primary and replicate connections for end-to-end replication from an Oracle primary to an Adaptive Server replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Oracle database with LogMiner
- two Replication Agent instances
- Replication Server
- a replicate Adaptive Server database

Task

1. At the primary Oracle database, select or create two tables that you want to replicate through two replication paths.
2. Use **rs_init** to add the replicate database to the replication system.
3. Create a default connection to the primary Oracle database:

```
create connection to pds.pdb
set error class rs_sqlserver_error_class
set function string class rs_oracle_function_class
set username muser
```

```
set password mpwd
with log transfer on,
dsi_suspended
go
```

where:

- *pds* is the value of the **rs_source_ds** parameter specified in Replication Agent.
- *pdb* is the value of **rs_source_db** specified in Replication Agent.
- *muser* is the maintenance user for the primary Oracle database.
- *mpwd* is the maintenance user password.

4. Create an alternate connection to the primary Oracle database:

```
create alternate connection to pds.pdb
named pds.conn2
set error class rs_sqlserver_error_class
set function string class rs_oracle_function_class
set username muser
set password mpwd
with primary only
go
```

5. Create a connection to the replicate Adaptive Server database:

```
create connection to rds.rdb
set error class rs_sqlserver_error_class
set function string class rs_sqlserver_function_class
set username muser
set password mpwd
go
```

where:

- *rds* is the name of the replicate Adaptive Server data server.
- *rdb* is the replicate Adaptive Server database.
- *muser* is the maintenance user for the replicate Adaptive Server database.
- *mpwd* is the maintenance user password.

6. Create an alternate connection to the replicate Adaptive Server database:

```
create alternate connection to rds.rdb
named rds.conn2
set error class rs_sqlserver_error_class
set function string class rs_sqlserver_function_class
set username muser
set password mpwd
go
```

7. Grant `create object` permission to **rs_username** on Replication Server:

```
grant create object to rs_username
go
```

where *rs_username* is the user login name that Replication Agent uses for Replication Server access.

8. For one instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri1
ra_config rs_source_ds, pds
ra_config rs_source_db, pdb
```

where

- `ra_user_1` is the user name used to create shared and instance objects in the primary database for use by Replication Agent instances. This user name must already be defined at the primary data server.
- `ra_` is the prefix used to identify share objects in the primary datbase. This prefix can be no longer than three characters.
- `ri1` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

9. Initialize the Replication Agent instance:

```
ra_admin init
```

10. For the other instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri2
ra_config rs_source_ds, pds
ra_config rs_source_db, conn2
```

where

- `ri2` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
- The values of **ra_admin_owner** and **ra_admin_prefix** are the same as all other Replication Agent instances in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

11. Initialize the Replication Agent instance:

```
ra_admin init
```

12. Mark the two tables from Step1 for replication. On the Replication Agent instance identified by `ri1`:

```
pdb_setreptable ptab1, mark
go
```

On the Replication Agent instance identified by `ri2`:

```
pdb_setreptable ptab2, mark
go
```

where `ptab1` and `ptab2` are the primary database tables to be replicated.

13. Create two replication definitions against the primary Oracle database:
    For example to create the `ptab1_repdef` replication definition for the `ptab1` table:

    ```
    create replication definition ptab1_repdef
    with primary at pds.pdb
    with all tables named 'ptab1'
    ...
    go
    ```

    To create the `ptab2_repdef` replication definition for the `ptab2` table:

    ```
    create replication definition ptab2_repdef
    with primary at pds.pdb
    with all tables named 'ptab2'
    ...
    go
    ```

    Note: These replication definitions must use the default primary connection name.

    If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

14. Resume the Replication Agent instances:

    ```
    resume
    ```

    If the Replication Agent instance fails to resume, verify that LogMiner is installed and configured. See the *Replication Agent Primary Database Guide > Replication Agent for Oracle > Oracle-Specific Considerations > Oracle Transaction and Operation Troubleshooting > Setting Up Replication Agent and Oracle to use ra_dumptran and ra_helpop*.

15. Create a subscription against the default replicate connection.
    For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

    ```
    create subscription ptab1_sub
    for ptab1_repdef
    with replicate at rds.rdb
    without materialization
    go
    ```

16. Create a subscription against the alternate replicate connection.
    For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

    ```
    create subscription ptab2_sub
    for ptab2_repdef
    with replicate at rds.conn2
    without materialization
    go
    ```

Multi-Path Replication from Oracle to HANA DB

Set up a Multi-Path Replication system with two primary and replicate connections for end-to-end replication from an Oracle primary to a HANA DB replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Oracle database with LogMiner
- two Replication Agent instances
- Replication Server
- ExpressConnect for HANA DB
- a replicate HANA DB database

Task

1. At the primary Oracle database, select or create two tables that you want to replicate through two replication paths.
2. Create a default connection to the primary Oracle database:

   ```
   create connection to pds.pdb
   set error class rs_sqlserver_error_class
   set function string class rs_oracle_function_class
   set username muser
   set password mpwd
   with log transfer on,
   dsi_suspended
   go
   ```

   where:
   - *pds* is the value of the **rs_source_ds** parameter specified in Replication Agent.
   - *pdb* is the value of **rs_source_db** specified in Replication Agent.
   - *muser* is the maintenance user for the primary Oracle database.
   - *mpwd* is the maintenance user password.
3. Create an alternate connection to the primary Oracle database:

   ```
   create alternate connection to pds.pdb
   named pds.conn2
   set error class rs_sqlserver_error_class
   set function string class rs_oracle_function_class
   set username muser
   set password mpwd
   with primary only
   go
   ```
4. At the replicate HANA DB instance, create a maintenance user for the replicate HANA DB instance:

   ```
   CREATE USER muser PASSWORD mpwd;
   ```

5. Grant these authorities to the maintenance user:
   - **CREATE ANY**
   - **DELETE**
   - **DROP**
   - **EXECUTE** (on all replicate stored procedures)
   - **INDEX**
   - **SELECT**
   - **UPDATE** (on all replicate tables)
6. If you are connecting to HANA DB with a standard connection and not using SAP Secure User Store, add an entry to your interfaces file identifying the replicate HANA DB data server, and restart Replication Server:

   ```
   [dataservername]
   master tcp ether hostname port
   query tcp ether hostname port
   ```

   where *hostname* and *port* are the host and port number of the HANA DB dataserver, and *dataservername* is a label used to identify the host and port number.

   If you are using SAP Secure User Store, create a user store of encrypted credentials:

   ```
   hdbuserstore set rds myhost:xxxxx my_securestore_user
   my_securestore_pwd
   ```

   where
   - *rds* is the key for the secure store entry
   - *myhost*:*xxxxx* is the connection environment host name and port number
   - *my_securestore_user* and *my_securestore_pwd* are SAP Secure User Store credentials
7. Create a connection to the replicate HANA DB instance using ExpressConnect for HANA DB.

   For a standard connection:

   ```
   create connection to rds.rdb
   using profile rs_ase_to_hanadb;ech
   set username muser
   set password mpwd
   go
   ```

   For SAP Secure User Store:

   ```
   create connection to rds.rdb
   using profile rs_ase_to_hanadb;ech
   set username auser
   set password apwd
   set dsi_connector_sec_mech to "hdbuserstore"
   go
   ```

   where:
   - *rds* is the replicate HANA DB data server. For a standard connection, this must match the interfaces file entry. For an SAP Secure User Store connection, this must match

what you used as the key to create a user store of encrypted credentials with the **hdbuserstore** utility.

- *rdb* is placeholder: You must provide a value, but it is not used..
- *muser* is the maintenance user for the replicate HANA DB instance.
- *mpwd* is the replicate HANA DB maintenance user password.
- *auser* and *apwd* are unused values supplied only to satisfy the syntax of the **create connection** command.

Note: The secure store must be created with the same operating system user ID that starts and runs Replication Server. Otherwise, Replication Server cannot access the secure user store.

8. Create an alternate connection to the replicate HANA DB database through ExpressConnect for HANA DB.

   For a standard connection:

   ```
   create alternate connection to rds.conn2
   using profile rs_ase_to_hanadb;ech
   set username muser
   set password mpwd
   go
   ```

   For SAP Secure User Store:

   ```
   create connection to rds.conn2
   using profile rs_ase_to_hanadb;ech
   set username auser
   set password apwd
   set dsi_connector_sec_mech to "hdbuserstore"
   go
   ```

9. Grant `create object` permission to **rs_username** on Replication Server:

   ```
   grant create object to rs_username
   go
   ```

   where *rs_username* is the user login name that Replication Agent uses for Replication Server access.

10. For one instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

    ```
    ra_config ra_admin_owner, ra_user_1
    ra_config ra_admin_prefix, ra_
    ra_config ra_admin_instance_prefix, ri1
    ra_config rs_source_ds, pds
    ra_config rs_source_db, pdb
    ```

    where

    - `ra_user_1` is the user name used to create shared and instance objects in the primary database for use by Replication Agent instances. This user name must already be defined at the primary data server.

- `ra_` is the prefix used to identify share objects in the primary datbase. This prefix can be no longer than three characters.
- `ri1` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

11. Initialize the Replication Agent instance:

```
ra_admin init
```

12. For the other instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri2
ra_config rs_source_ds, pds
ra_config rs_source_db, conn2
```

where

- `ri2` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
- The values of **ra_admin_owner** and **ra_admin_prefix** are the same as all other Replication Agent instances in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

13. Initialize the Replication Agent instance:

```
ra_admin init
```

14. Mark the two tables from Step1 for replication. On the Replication Agent instance identified by `ri1`:

```
pdb_setreptable ptab1, mark
go
```

On the Replication Agent instance identified by `ri2`:

```
pdb_setreptable ptab2, mark
go
```

where `ptab1` and `ptab2` are the primary database tables to be replicated.

15. Create two replication definitions against the primary Oracle database:
For example to create the `ptab1_repdef` replication definition for the `ptab1` table:

```
create replication definition ptab1_repdef
with primary at pds.pdb
with all tables named 'ptab1'
...
go
```

To create the `ptab2_repdef` replication definition for the `ptab2` table:

```
create replication definition ptab2_repdef
with primary at pds.pdb
with all tables named 'ptab2'
...
go
```

Note: These replication definitions must use the default primary connection name.

If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

16. Resume the Replication Agent instances:

```
resume
```

If the Replication Agent instance fails to resume, verify that LogMiner is installed and configured. See the *Replication Agent Primary Database Guide > Replication Agent for Oracle > Oracle-Specific Considerations > Oracle Transaction and Operation Troubleshooting > Setting Up Replication Agent and Oracle to use ra_dumptran and ra_helpop*.

17. Create a subscription against the default replicate connection.
For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

```
create subscription ptab1_sub
for ptab1_repdef
with replicate at rds.rdb
without materialization
go
```

18. Create a subscription against the alternate replicate connection.
For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

```
create subscription ptab2_sub
for ptab2_repdef
with replicate at rds.conn2
without materialization
go
```

Multi-Path Replication from Oracle to Oracle

Set up a Multi-Path Replication system with two primary and replicate connections for end-to-end replication from an Oracle primary to an Oracle replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Oracle database with LogMiner
- two Replication Agent instances
- Replication Server
- ExpressConnect for Oracle

- a replicate Oracle database

Task

1. At the primary Oracle database, select or create two tables that you want to replicate through two replication paths.
2. Create a default connection to the primary Oracle database:

   ```
   create connection to pds.pdb
   set error class rs_sqlserver_error_class
   set function string class rs_oracle_function_class
   set username muser
   set password mpwd
   with log transfer on,
   dsi_suspended
   go
   ```

   where:
   - *pds* is the value of the **rs_source_ds** parameter specified in Replication Agent.
   - *pdb* is the value of **rs_source_db** specified in Replication Agent.
   - *muser* is the maintenance user for the primary Oracle database.
   - *mpwd* is the maintenance user password.
3. Create an alternate connection to the primary Oracle database:

   ```
   create alternate connection to pds.pdb
   named pds.conn2
   set error class rs_sqlserver_error_class
   set function string class rs_oracle_function_class
   set username muser
   set password mpwd
   with primary only
   go
   ```
4. Copy the `tnsnames.ora` file for your replicate Oracle database to the Replication Server `RS_installation_directory\ REP-15_5\connector\oraoci \network\admin` directory, and restart Replication Server.
5. Create a connection to the replicate Oracle database through ExpressConnect for Oracle.

   ```
   create connection to tns_alias_name.rdb
   using profile rs_oracle_to_oracle;eco
   set username muser
   set password mpwd
   set dsi_dataserver_make to 'ora'
   set dsi_connector_type to 'oci'
   set batch to 'off'
   go
   ```

   where:
   - *tns_alias_name* is the alias name for the replicate Oracle database defined in the `tnsnames.ora` file for the replicate Oracle database.

- *rdb* is the replicate Oracle System ID (SID) that is paired with the above **tns_alias_name** in the `tnsnames.ora` file. The default value is ORCL.
- *muser* is the maintenance user for the replicate Oracle database.
- *mpwd* is the replicate Oracle maintenance user password.

See the *ExpressConnect for Oracle Installation and Configuration Guide* for more information about naming the default connection.

6. Create an alternate connection to the replicate Oracle database through ExpressConnect for Oracle.

```
create alternate connection to tns_alias_name.rdb
named tns_alias_name.conn2
set error class rs_oracle_error_class
set function string class rs_oracle_function_class
set username muser
set password mpwd
set dsi_dataserver_make to 'ora'
set dsi_dataserver_type to 'oci'
set batch to 'off'
set dsi_proc_as_rpc to 'on'
go
```

7. Grant `create object` permission to **rs_username** on Replication Server:

```
grant create object to rs_username
go
```

where *rs_username* is the user login name that Replication Agent uses for Replication Server access.

8. For one instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri1
ra_config rs_source_ds, pds
ra_config rs_source_db, pdb
```

where

- `ra_user_1` is the user name used to create shared and instance objects in the primary database for use by Replication Agent instances. This user name must already be defined at the primary data server.
- `ra_` is the prefix used to identify share objects in the primary datbase. This prefix can be no longer than three characters.
- `ri1` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

9. Initialize the Replication Agent instance:

```
ra_admin init
```

10. For the other instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri2
ra_config rs_source_ds, pds
ra_config rs_source_db, conn2
```

    where

    - `ri2` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
    - The values of **ra_admin_owner** and **ra_admin_prefix** are the same as all other Replication Agent instances in the replication group.
    - The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

11. Initialize the Replication Agent instance:

```
ra_admin init
```

12. Mark the two tables from Step1 for replication. On the Replication Agent instance identified by `ri1`:

```
pdb_setreptable ptab1, mark
go
```

    On the Replication Agent instance identified by `ri2`:

```
pdb_setreptable ptab2, mark
go
```

    where `ptab1` and `ptab2` are the primary database tables to be replicated.

13. Create two replication definitions against the primary Oracle database:
    For example to create the `ptab1_repdef` replication definition for the `ptab1` table:

```
create replication definition ptab1_repdef
with primary at pds.pdb
with all tables named 'ptab1'
...
go
```

    To create the `ptab2_repdef` replication definition for the `ptab2` table:

```
create replication definition ptab2_repdef
with primary at pds.pdb
with all tables named 'ptab2'
...
go
```

Note: These replication definitions must use the default primary connection name.

If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

14. Resume the Replication Agent instances:

```
resume
```

If the Replication Agent instance fails to resume, verify that LogMiner is installed and configured. See the *Replication Agent Primary Database Guide > Replication Agent for Oracle > Oracle-Specific Considerations > Oracle Transaction and Operation Troubleshooting > Setting Up Replication Agent and Oracle to use ra_dumptran and ra_helpop*.

15. Create a subscription against the default replicate connection.
For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

```
create subscription ptab1_sub
for ptab1_repdef
with replicate at tns_alias_name.rdb
without materialization
go
```

16. Create a subscription against the alternate replicate connection.
For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

```
create subscription ptab2_sub
for ptab2_repdef
with replicate at tns_alias_name.conn2
without materialization
go
```

See also

- *Oracle as Replicate Data Server* on page 103

Multi-Path Replication from Oracle to Sybase IQ

Set up a Multi-Path Replication system with two primary and replicate connections for end-to-end replication from an Oracle primary to a Sybase IQ replicate.

Prerequisites

This scenario assumes you have already installed and configured:

- a primary Oracle database with LogMiner
- two Replication Agent instances
- Replication Server
- a replicate Sybase IQ database

Task

1. At the primary Oracle database, select or create two tables that you want to replicate through two replication paths.
2. Create a default connection to the primary Oracle database:

```
create connection to pds.pdb
set error class rs_sqlserver_error_class
set function string class rs_oracle_function_class
set username muser
set password mpwd
with log transfer on,
dsi_suspended
go
```

   where:
   - *pds* is the value of the **rs_source_ds** parameter specified in Replication Agent.
   - *pdb* is the value of **rs_source_db** specified in Replication Agent.
   - *muser* is the maintenance user for the primary Oracle database.
   - *mpwd* is the maintenance user password.
3. Create an alternate connection to the primary Oracle database:

```
create alternate connection to pds.pdb
named pds.conn2
set error class rs_sqlserver_error_class
set function string class rs_oracle_function_class
set username muser
set password mpwd
with primary only
go
```

4. Create a connection to the replicate Sybase IQ database using a connection profile:

```
create connection to rds.rdb
using profile rs_oracle_to_iq;standard
set username muser
set password mpwd
go
```

   where:
   - *rds* is the name of the replicate Sybase IQ server.
   - *rdb* is the replicate Sybase IQ database.
   - *muser* is the maintenance user for the replicate Sybase IQ database.
   - *mpwd* is the replicate Sybase IQ maintenance user password.
5. Create an alternate connection to the replicate Sybase IQ database:

```
create alternate connection to rds.rdb
named rds.conn2
using profile rs_oracle_to_iq;standard
set username muser2
set password mpwd
go
```

- You must use a different maintenance user name for each alternate connection you create to a Sybase IQ database. Ensure that you use unique maintenance user names even if you are creating connections from different Replication Servers to a Sybase IQ database.
- If you use a different name for the replicate Sybase IQ server in the alternate connection than the one you used previously, you must have an entry for the different name in the Replication Server interfaces file.

6. Grant `create object` permission to **rs_username** on Replication Server:

```
grant create object to rs_username
go
```

   where *rs_username* is the user login name that Replication Agent uses for Replication Server access.

7. For one instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri1
ra_config rs_source_ds, pds
ra_config rs_source_db, pdb
```

   where

   - `ra_user_1` is the user name used to create shared and instance objects in the primary database for use by Replication Agent instances. This user name must already be defined at the primary data server.
   - `ra_` is the prefix used to identify share objects in the primary datbase. This prefix can be no longer than three characters.
   - `ri1` is the prefix that uniquely identifies this Replication Agent instance in the replication group.
   - The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

8. Initialize the Replication Agent instance:

```
ra_admin init
```

9. For the other instance of Replication Agent, set the **ra_admin_owner**, **ra_admin_prefix**, **ra_admin_instance_prefix**, **rs_source_ds**, and **rs_source_db** parameters:

```
ra_config ra_admin_owner, ra_user_1
ra_config ra_admin_prefix, ra_
ra_config ra_admin_instance_prefix, ri2
ra_config rs_source_ds, pds
ra_config rs_source_db, conn2
```

   where

   - `ri2` is the prefix that uniquely identifies this Replication Agent instance in the replication group.

- The values of **ra_admin_owner** and **ra_admin_prefix** are the same as all other Replication Agent instances in the replication group.
- The value of **rs_source_ds** combined with the value of **rs_source_db** forms the connection name that this Replication Agent instance uses to connect to Replication Server.

10. Initialize the Replication Agent instance:

```
ra_admin init
```

11. Mark the two tables from Step1 for replication. On the Replication Agent instance identified by `ri1`:

```
pdb_setreptable ptab1, mark
go
```

On the Replication Agent instance identified by `ri2`:

```
pdb_setreptable ptab2, mark
go
```

where `ptab1` and `ptab2` are the primary database tables to be replicated.

12. Create two replication definitions against the primary Oracle database:
For example, to create the `ptab1_repdef` replication definition for the `ptab1` table:

```
create replication definition ptab1_repdef
with primary at pds.pdb
with all tables named 'ptab1'
...
go
```

To create the `ptab2_repdef` replication definition for the `ptab2` table:

```
create replication definition ptab2_repdef
with primary at pds.pdb
with all tables named 'ptab2'
...
go
```

Note: These replication definitions must use the default primary connection name.

If the primary connection and alternate primary connection are on different Replication Servers, create replication definitions on each Replication Server.

13. Resume the Replication Agent instances:

```
resume
```

If the Replication Agent instance fails to resume, verify that LogMiner is installed and configured. See the *Replication Agent Primary Database Guide > Replication Agent for Oracle > Oracle-Specific Considerations > Oracle Transaction and Operation Troubleshooting > Setting Up Replication Agent and Oracle to use ra_dumptran and ra_helpop*.

14. Create a subscription against the default replicate connection..
For example, to create the `ptab1_sub` subscription for the `ptab1_repdef` replication definition:

```
create subscription ptab1_sub
for ptab1_repdef
with replicate at rds.rdb
without materialization
go
```

15. Create a subscription against the alternate replicate connection.
    For example, to create the `ptab2_sub` subscription for the `ptab2_repdef` replication definition:

```
create subscription ptab2_sub
for ptab2_repdef
with replicate at rds.conn2
without materialization
go
```

See also

- *Sybase IQ as Replicate Data Server* on page 113

Heterogeneous Warm Standby for Oracle

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

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 a warm standby application.

In the 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.

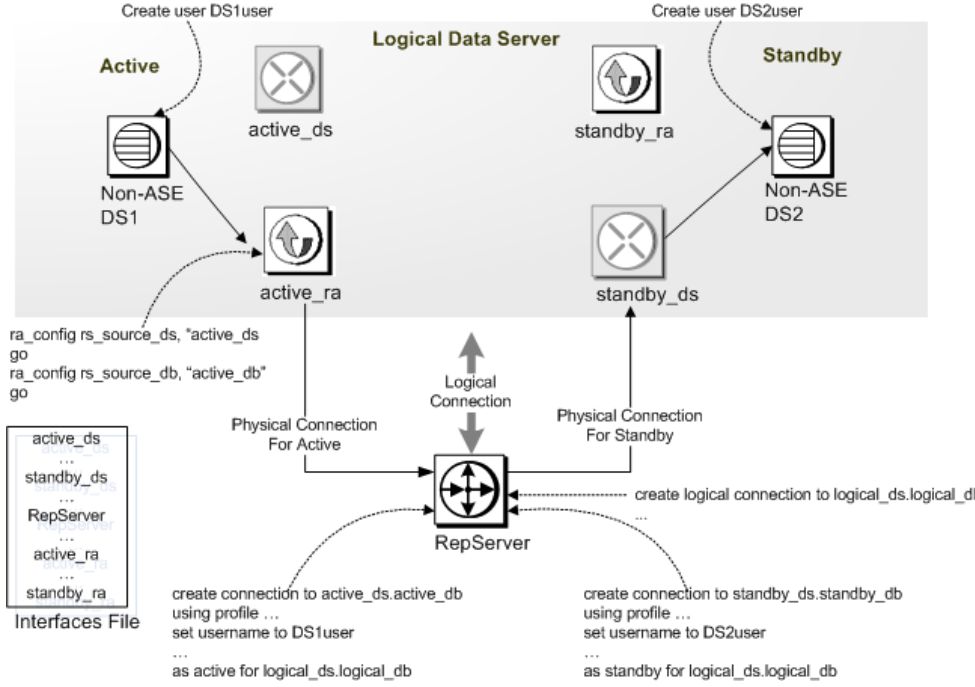
In many Replication Server applications:

- 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 *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications*.

Warm Standby Application

Illustrates the normal operation of an example warm standby application.



Warm Standby Requirements and Restrictions

Be familiar with the requirements and restrictions of 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.
- Replication Server does not support warm standby replication between different platforms.
- 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.

The function string class includes:

- **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.

Replicated Information 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 *Replication Agent Reference Manual > Configuration Parameters*.

Setting Up Warm Standby Databases

Set up databases for a warm standby application.

Prerequisites

Perform these tasks 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 databases.
- 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 databases.
- Define the DDL user name in both the active and the standby databases, 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

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.

Creating the Logical Connection

Create the logical connection for a warm standby application.

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:

   logical_ds.logical_db

   ```
   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.

Naming the Logical Connection

Use the form *logical_ds.logical_db* to name the logical connection.

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.

Initializing the 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 server 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
```

Adding the Active Database to the Replication System

Create 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
```

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, see the Oracle documentation.

Initializing the Replication Agent for the Standby Database

Start the Replication Agent for Oracle (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 server 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

Create a standby database connection.

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

Resume the active database and standby database connections.

During initialization of the standby database, Replication Server suspended the connection to the active database.

1. Execute this command in the Replication Server:

```
resume connection to active_ds.active_db
```

2. 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.

1. At the Replication Server, enter:

   ```
   switch active for logical_ds.logical_db
   to standby_ds.standby_db
   ```

   See "Internal Switching Steps" 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).

See also

- *Internal Switching Steps* on page 212

Before Switching Active and Standby Databases

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.

It represents the warm standby application in normal operation, before you switch the active and standby databases.

Figure 12: Warm Standby Application—Before Switching

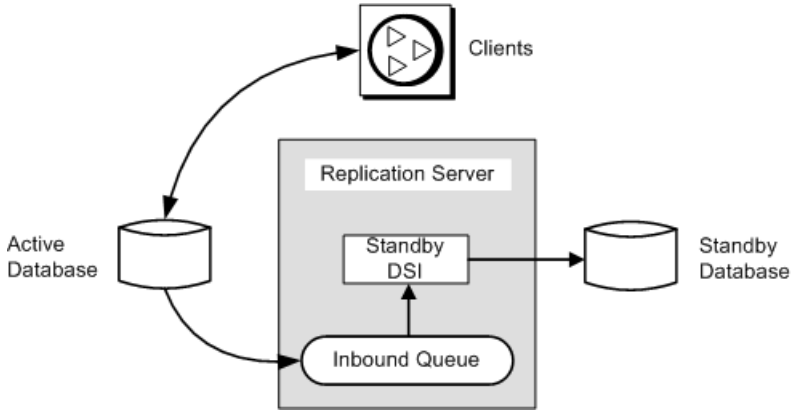


Figure: Warm Standby Application—Before Switching includes internal details to show that:

- 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

Learn about the 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.

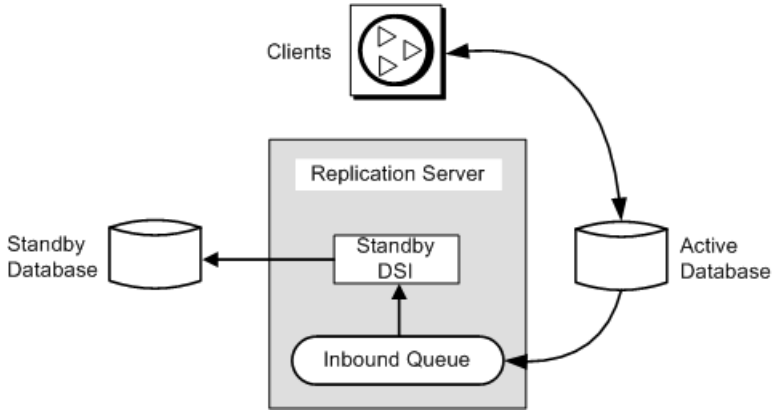
5. Stops the connection for the new active database. Purges the inbound queue and flushes the last oqid to `rs_oqid` table and resets `rs_locator` table accordingly. Resets the Replication Server segments flag and suspends the new standby DSI.
6. Updates the **ptype** parameter for both new active database 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 Active and Standby Databases

Learn the processes involved and the status of the components in a warm standby environment after you switch from the active to the standby database.

After you have switched the roles of the active and standby databases, the replication system will have changed, as shown in this figure:

Figure 13: Warm Standby Application Example—After Switching



- The previous standby database is the new active database. Client applications will 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.

Note: After switching, the Replication Agent for the previous active database has shut down, and the Replication Agent for the new active database has started.

Warm Standby Application Monitoring

You can monitor warm standby applications using Replication Server log file or 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 *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications*.

Replication Definitions and Subscriptions

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 for Oracle warm standby.

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.

Additional Replication Definitions for Warm Standby Databases

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.

The replication definitions that Replication Agent automatically created at initialization 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.

These are the cases in which you may want to create additional replication definitions:

- Provide mappings from Oracle datatypes to Replication Server UDD.
- 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.

See also

- *Datatype Translation and Mapping* on page 229

Subscriptions with Warm Standby Applications

The **create subscription** and **define subscription** commands use the logical database and data server names instead of the physical names.

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.

For detailed information about warm standby applications using replication, see *Replication Server Administration Guide Volume 2 > Managing Warm Standby Applications*.

Upgrade Considerations

Oracle warm standby feature does not require any special instructions that you may need to perform after an upgrade.

See *Replication Server Configuration Guide > Upgrade or Downgrade Replication Server* for your platform for information on upgrading your Replication Server version.

Downgrade Considerations

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.

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, thus you need to resume the replication process.

Resuming Replication After Downgrade

Learn to resume replication after downgrading the RSSD.

To resume 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.

Oracle Replicate Databases Resynchronization

Replication Server allows you to resynchronize and materialize the replicate database, and resume further replication without loss or a risk of inconsistent 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

Use the versions of Oracle, Replication Agent for Oracle, and ExpressConnect for Oracle that support the resynchronization of Oracle databases.

See the Replication Server Options documentation.

Table 3. Product Compatibility for Resynchronizing Oracle Databases

Database Server Version	Replication Agent Version	ExpressConnect Version
Oracle Server 10g, 11g	15.5	ExpressConnect 15.5 for Oracle

Configuring Database Resynchronization

Set up Oracle databases resynchronization.

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. Reinitialize the replicate database.
7. Resume replication.

You must use commands and parameters from both Replication Server and Replication Agent for Oracle for database resynchronization.

See also

- *Database Resynchronization Scenarios* on page 223

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 *Replication Server Reference Manual > Replication Server Commands > **resume connection***.

Run this command:

```
resume connection to data_server.database
   [skip [n] transaction | execute transaction | skip to resync
marker]
```

Warning! If you execute **resume connection** with the **skip to resync marker** option on the wrong connection, data on the replicate database becomes unsynchronized.

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**.

Send 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 *Replication Agent Reference Manual > Command Reference >* ***resume***.

Send 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.

Send 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 also

- *Resynchronizing Both the Primary and Replicate Databases from the Same Dump* on page 227

Obtain a Dump of the Database

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.

In Oracle, use **backup database plus archivelog** to dump the primary database, and **restore database** and **recover database** to apply the dump to the replicate database. 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**:

```
RMAN>list backup;
List of Backup Sets
===================


BS Key  Size       Device Type Elapsed Time Completion Time
------- ---------- ----------- ------------ ---------------
8       125.58M    DISK        00:00:04     16-MAY-11
        BP Key: 8   Status: AVAILABLE  Compressed: NO  Tag:
TAG20110516T125049
        Piece Name: /ralinuxsh5/oracle/product/11.1.0/db_2/dbs/
0bmcflp9_1_1

  List of Archived Logs in backup set 8
  Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
  ---- ------- ---------- --------- ---------- ---------
  1    1       1018110    14-MAY-11 1058201    15-MAY-11
  1    2       1058201    15-MAY-11 1103370    15-MAY-11
  1    3       1103370    15-MAY-11 1142662    16-MAY-11
  1    4       1142662    16-MAY-11 1148674    16-MAY-11
  1    5       1148674    16-MAY-11 1150375    16-MAY-11
  1    6       1150375    16-MAY-11 1150477    16-MAY-11

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
9       Full    1.08G      DISK        00:00:15     16-MAY-11
        BP Key: 9   Status: AVAILABLE  Compressed: NO  Tag:
TAG20110516T125054
        Piece Name: /ralinuxsh5/oracle/product/11.1.0/db_2/dbs/
0cmcflpe_1_1
  List of Datafiles in backup set 9
```

```
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  1      Full 1150485   16-MAY-11 /work2/oracle11.1/oradata/or11sh1/
system01.dbf
  2      Full 1150485   16-MAY-11 /work2/oracle11.1/oradata/or11sh1/
sysaux01.dbf
  3      Full 1150485   16-MAY-11 /work2/oracle11.1/oradata/or11sh1/
undotbs01.dbf
  4      Full 1150485   16-MAY-11 /work2/oracle11.1/oradata/or11sh1/
users01.dbf
  5      Full 1150485   16-MAY-11 /work2/oracle11.1/oradata/or11sh1/
example01.dbf

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
10      Full    9.36M      DISK        00:00:04     16-MAY-11
        BP Key: 10   Status: AVAILABLE  Compressed: NO  Tag:
TAG20110516T125054
        Piece Name: /ralinuxsh5/oracle/product/11.1.0/db_2/dbs/
0dmcflq1_1_1
  SPFILE Included: Modification time: 14-MAY-11
  SPFILE db_unique_name: OR11SH1
  Control File Included: Ckp SCN: 1150507      Ckp time: 16-MAY-11

BS Key  Size       Device Type Elapsed Time Completion Time
------- ---------- ----------- ------------ ---------------
11      18.50K     DISK        00:00:04     16-MAY-11
        BP Key: 11   Status: AVAILABLE  Compressed: NO  Tag:
TAG20110516T125118
        Piece Name: /ralinuxsh5/oracle/product/11.1.0/db_2/dbs/
0emcflq6_1_1

  List of Archived Logs in backup set 11
  Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
  ---- ------- ---------- --------- ---------- ---------
  1    7       1150477    16-MAY-11 1150513    16-MAY-11
  1    8       1150513    16-MAY-11 1150568    16-MAY-11
```

The required dump point is one substracted from the maximum value in the "Next SCN" column in the last Archived Logs backup set. In this example, the last set is set 11 and the maximum value for "Next SCN" in set 11 is 1150568. Therefore, the dump point in this example is 1150567.

See the Oracle documentation for more information on the RMAN utility and the **list backup** command.

Send 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.

Monitor DSI Thread Information

Use the **admin who** command to provide information on DSI during database resynchronization.

State	Description
SkipUntil Resync	DSI resumes after you execute **skip to resync**. This state remains until DSI receives a resync database marker.
SkipUntil Dump	DSI has received a resync database marker. This state remains until DSI has processed a subsequent dump database marker.

Apply the Dump to a Database to be Resynchronized

You can load the primary database dump to the replicate database only after you see messages in the system log.

These are the messages:

- 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 the Oracle documentation for instructions to load the dump to the database you want to resynchronize.

Reinitializing the Replicate Database

After you apply the dump from the primary database or dump source to the replicate database, reinitialize the replicate database to restore users, tables, and permissions that the dump removed.

1. If maintenance and DDL users do not exist in the primary database, add them to the replicate database after you apply the dump from the primary database.
2. Run the `hds_oracle_new_setup_for_replicate.sql` script on the replicate database to add the relevant Replication Server system tables to the replicate database.

   The script also inserts relevant values and grants the required permissions in the replicate database.

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*.

Resynchronize One or More Replicate Databases Directly from a Primary Database

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.

- Coordinate resynchronization if the primary site is a logical database that consists of a warm standby pair of databases that you want to resynchronize with one or more replicate databases. In a warm standby pair, the active database acts as the primary database, and the standby acts as the replicate database. Therefore, the active database of a warm standby pair at a primary site also appears as a primary database to one or multiple replicate databases.

Resynchronizing Directly from a Primary Database

Resynchronize a replicate database from a primary database.

1. To move the truncation point, execute:

   ```
   ra_locator move_truncpt
   ra_admin refresh
   isql -USAMPLE_RS_RSSD_prim -PSAMPLE_RS_RSSD_prim_ps
   -SSAMPLE_RS_ERSSD -DSAMPLE_RS_ERSSD
   rs_zeroltm NY, NYora92
   go
   ```

   If the truncation point is not changed, proceed to step 2.
2. Stop replication processing by Replication Agent. In Replication Agent, execute:

   ```
   suspend
   ```
3. Suspend the Replication Server DSI connection to the replicate database:

   ```
   suspend connection to dataserver.database
   ```
4. 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
   ```
5. Obtain a dump of the primary database contents following the instructions in your database documentation.
6. If you use the Recovery Manager (RMAN) for Oracle, use the Oracle **list backup** command to obtain the last System Change Number (SCN) of the RMAN backup. Then, in Replication Agent, set this SCN as the value of **lr_dump_marker**:

   ```
   lr_dump_marker oracle scn
   ```
7. Start your Replication Agent in resync mode and send a resync marker to Replication Server:

   ```
   resume resync
   go
   ```
8. 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 database you are resynchronizing has accepted the resync marker.

9. Apply the dump of the primary database to the replicate database following the instructions in your database 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. If the maintenance and DDL users do not exist in the primary database, add these users to the replicate database after you apply the dump from the primary database.
12. Run the `hds_oracle_new_setup_for_replicate.sql` script on the replicate database to add the `rs_info` and `rs_lastcommit` tables to the replicate database.

    The script also inserts relevant values and grants the required permissions in the replicate database.
13. 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

Coordinate resynchronization after you dump the primary database using a third-party **dump** utility, such as a 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.

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. If the truncation point has not been moved, proceed to step 5. Otherwise, reinitialize the Replication Agent repository before you obtain a dump of the primary database contents. In the Replication Agent, execute:

```
ra_init force
go
```

5. Use the third-party utility to obtain a dump of the primary database contents.
6. Determine the dump point based on information from the primary database when you took the dump, or information from the third-party utility. With a third-party utility, 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 in-progress transactions from the disk snapshot, and then use the “end of transaction log” point as the dump database marker.
7. To mark the end of the dump position that you obtained in step 5, execute the stored procedure on the primary database for Replication Agent:

   ```
   lr_dump_marker oracle scn
   ```
8. Restart Replication Agent in resync mode and send a resync marker to Replication Server:

   ```
   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 6 and set in step 7, and sends the dump database marker to Replication Server.
9. 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.
   ```
10. Apply the dump of the primary database from the third-party tool to the replicate database, following the instructions in the database and third-party utility documentation.
11. 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.
12. If the maintenance and DDL users do not exist in the primary database, add these users to the replicate database after you apply the dump from the primary database.
13. Run the `hds_oracle_new_setup_for_replicate.sql` script on the replicate database to add the `rs_info` and `rs_lastcommit` tables to the replicate database.

    The script also inserts relevant values and grants the required permissions in the replicate database.
14. 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

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.

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. If the maintenance and DDL users do not exist in the primary database, add these users to the replicate database after you apply the dump from the primary database.
11. Run the `hds_oracle_new_setup_for_replicate.sql` script on the replicate database to add the `rs_info` and `rs_lastcommit` tables to the replicate database.

The script also inserts relevant values and grants the required permissions in the replicate database.

12. 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
```

Datatype Translation and Mapping

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.

Homogeneous Mapping

Replication Server supports datatype translation and mapping between data servers of the same type and version.

Default Mapping

Replication Server supports replication between data servers of the same type and version for all supported datatypes. By default, Replication Server replicates data of all datatypes captured at the primary database to the same datatype on the replicate database.

User-Defined Mapping

For replication between data servers of the same type and version, you can override the default Replication Server mapping and attempt to replicate from one datatype at the primary database to a different datatype at the replicate database by:

- altering the published datatype for a column in the replication definition, or
- using custom function strings

Note: Be sure you fully understand the capabilities of these customization features and your business objectives before overriding the default datatype mapping behavior.

One way to alter the published datatype is by using the **map to** clause in a replication definition. See *Replication Server Administration Guide Volume 1 > Getting Started with Translating Datatypes > Create Column-Level Translations*.

Heterogeneous Mapping

Replication Server supports datatype translation and mapping between data servers of a different type or version.

Replication Server provides default support for datatypes on the primary database that have a datatype of matching length, scale, range and interpretation on the replicate database. Primary datatypes that have no match on the replicate database are supported but require customization.

Default Mapping

If there is no difference between the primary and replicate datatype—as in mapping Oracle `CHAR` to HANA DB `char`—or if the replicate datatype exceeds the limits of the primary—as in mapping Oracle `CHAR` to HANA DB `varchar`—Replication Server makes no special adjustment.

If the only difference between the primary and replicate datatype is in semantics or scale—as in mapping Adaptive Server `datetime` to HANA DB `timestamp`—Replication Server performs default class-level translations to match the primary datatype value to that of the replicate database.

Custom Mapping

For replication between data servers of a different type or version, Replication Server provides no default mapping for primary datatypes that have no match on the replicate database. You must provide custom mapping if:

- The length, precision, or range of the replicate datatype is narrower than that of the primary datatype and may result in the truncation of data.
- The replicate data server has no datatype with the same purpose as that of the primary datatype—for example, intervals, sequences, XML types, or spatial types on the primary have no equivalent datatypes on the replicate data server.

You can customize the Replication Server mapping and attempt to replicate from one datatype at the primary database to a different datatype at the replicate database by:

- altering the published datatype for a column in the replication definition, or
- using custom function strings

Note: Be sure you fully understand the capabilities of these customization features and your business objectives before customizing the mapping behavior.

One way to alter the published datatype is by using the **map to** clause in a replication definition. For example, to map an Oracle column named `col2` of type `INTERVAL` to a HANA DB column of type `varchar`:

```
col2 rs_oracle_interval map to varchar
```

To preview the expected results of this mapping, use the Replication Server **admin translate** command:

```
1> admin translate, '+000000005 01:01:01.000000000',
rs_oracle_interval, 'varchar(40)'
2> go
```

```
Delimiter Prefix  Translated Value               Delimiter Postfix
----------------  ----------------------------  -----------------
'                 +000000005 01:01:01.000000000  '
```

See *Replication Server Administration Guide Volume 1 > Getting Started with Translating Datatypes > Create Column-Level Translations*.

DB2 Datatypes

Information about datatype translation applies to DB2 UDB in either mainframe environments (such as IBM z/OS), or UNIX and Microsoft Windows environments..

Adaptive Server to DB2 Datatypes

Lists class-level translations from Adaptive Server datatypes 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

Adaptive Server Datatype	DB2 Datatype
smallmoney	NUMERIC
time	TIME (UNIX and Windows only)
tinyint	NUMERIC
unsigned bigint	DECIMAL (20,0)
unsigned int	BIGINT
unsigned smallint	INTEGER
unsigned tinyint	SMALLINT
unitext	DBCLOB
varbinary	VARCHAR FOR BIT DATA

DB2 to Adaptive Server Datatypes

Lists class-level translations from DB2 datatypes 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 HANA DB

Lists DB2 UDB-to-HANA DB datatype mappings.

See *SAP HANA Reference > SQL Reference Manual > Data Types* for current information on the range of values that can be represented by HANA DB datatypes.

Table 4. DB2 UDB-to-HANA DB Datatype Mappings

DB2 UDB Datatype	Replication Definition Datatype	HANA Datatype
BIGINT	rs_udb_bigint	bigint
BLOB	image	blob
CHAR	char	char (See note below.)
CHAR FOR BIT DATA	rs_udb_char_for_bit (up to 255 bytes) varbinary (greater than 255 bytes)	varbinary
CLOB	text	clob
DATE	rs_udb_date	date
DBCLOB	unitext	nclob
DECFLOAT(16)	rs_udb_decfloat	float
DECFLOAT(32)	rs_udb_decfloat	decimal
DECIMAL	decimal	decimal
DOUBLE	rs_udb_double	double
GRAPHIC	unichar	nchar
INTEGER	integer	integer
LONG VARCHAR	text	clob
LONG VARCHAR FOR BIT DATA	image	blob
LONG VARGRAPHIC	unitext	nclob
REAL	rs_udb_real	real
SMALLINT	smallint	smallint
TIME	rs_udb_time	time
TIMESTAMP(12)	rs_udb_time-stamp12	timestamp

DB2 UDB Datatype	Replication Definition Datatype	HANA Datatype
VARCHAR	varchar	varchar (See note below.)
VARCHAR FOR BIT DATA	rs_udb_var-char_for_bit (up to 255 bytes) varbinary (greater than 255 bytes)	varbinary
VARGRAPHIC	univarchar	nvarchar

Note: If your primary database is configured for a single-byte character set (like iso_1) that accepts extended ASCII characters—the one-byte ASCII character values between 128 and 255—HANA DB may convert single-byte extended characters into double-byte characters in char or varchar columns. This conversion may subsequently cause the storage size in HANA DB to exceed the original column length or storage of the primary database. If your primary database and application support using extended ASCII characters in a single-byte character set, you may need to increase the size of the corresponding column in the HANA DB table definition. This change allows those columns to accommodate the conversion of extended characters to two-byte storage.

DB2 to Microsoft SQL Server Datatypes

Lists class-level translations from DB2 datatypes 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

Lists class-level translations from DB2 datatypes 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

Lists the Replication Server user-defined datatype (UDD) names that identify DB2 datatypes for DB2 data servers on z/OS platforms.

Table 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*

Lists the Replication Server UDD names that identify DB2 datatypes for DB2 data servers on UNIX and Microsoft Windows platforms.

Table 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*

DB2 UNIX and Windows Datatypes	Replication Server Name
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

Learn about the class-level translations (default datatype mapping) for Microsoft SQL Server datatypes and Replication Server datatype names for Microsoft SQL Server datatypes.

Adaptive Server to Microsoft SQL Server Datatypes

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 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 Datatype

Lists class-level translations from Microsoft SQL Server datatypes to DB2 datatypes.

Table 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 HANA DB

Lists Microsoft SQL Server-to-HANA DB datatype mappings.

See *SAP HANA Reference > SQL Reference Manual > Data Types* for current information on the range of values that can be represented by HANA DB datatypes.

Table 9. Microsoft SQL Server-to-HANA DB Datatype Mappings

Microsoft SQL Server Datatype	Replication Definition Datatype	HANA DB Datatype
bigdatetime	bigdatetime	timestamp
bigint	bigint	bigint
bigtime	bigtime	No default
binary	binary	varbinary
bit	tinyint	tinyint
char	char	char (See note below.)
date	date	date

Microsoft SQL Server Datatype	Replication Definition Datatype	HANA DB Datatype
datetime	datetime	timestamp
decimal	decimal varchar	decimal (to 34 digits) varchar (over 34 digits)
double precision	float	double
float	float	float
identity	numeric	numeric
int	integer	integer
integer	rs_msss_bigint	bigint
image	image	blob
longsysname varchar(255) Not Null	varchar(255)	varchar(255)
money	decimal	decimal(19,4)
nchar	char	nchar
numeric	numeric	decimal
nvarchar	varchar	nvarchar
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	decimal	decimal(10,4)
sysname varchar(30) Not Null	varchar(30)	varchar(30)
text	text	clob
timestamp	timestamp	No default
tinyint	tinyint	tinyint
time	time	time
unichar	unichar	nchar

Microsoft SQL Server Datatype	Replication Definition Datatype	HANA DB Datatype
univarchar	univarchar	nvarchar
unsigned bigint		decimal
unsigned int	unsigned int	bigint
unsigned smallint	unsigned smallint	integer
unsigned bigint	unsigned bigint	decimal
unitext	unitext	nclob
varbinary	varbinary	varbinary
varchar	varchar	varchar (See note below.)

Note: If your primary database is configured for a single-byte character set (like iso_1) that accepts extended ASCII characters—the one-byte ASCII character values between 128 and 255—HANA DB may convert single-byte extended characters into double-byte characters in `char` or `varchar` columns. This conversion may subsequently cause the storage size in HANA DB to exceed the original column length or storage of the primary database. If your primary database and application support using extended ASCII characters in a single-byte character set, you may need to increase the size of the corresponding column in the HANA DB table definition. This change allows those columns to accommodate the conversion of extended characters to two-byte storage.

Microsoft SQL Server to Oracle Datatypes

Lists class-level translations from Microsoft SQL Server datatypes to Oracle datatypes.

Table 10. 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

Lists the Replication Server user-defined datatype (UDD) name that identifies a Microsoft SQL Server datatype

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 11. Replication Server Names for Microsoft SQL Server Datatypes

Microsoft SQL Server Datatype	Replication Server Name
integer	*rs_msss_bigint*

Oracle Datatypes

Learn about the class-level translations (default datatype mapping) for Oracle datatypes and Replication Server datatype names for Oracle datatypes.

Adaptive Server to Oracle Datatypes

Lists class-level translations from Adaptive Server datatypes to Oracle datatypes.

Table 12. 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
smallmoney	DECIMAL
time	DATE (with time)
unsigned tinyint	SMALLINT
unsigned smallint	INTEGER

Adaptive Server Datatype	Oracle Datatype
unsigned int	NUMBER
unsigned bigint	NUMBER
unitext	NCLOB
varbinary	RAW

Oracle to Adaptive Server Datatypes

Lists class-level translations from Oracle datatypes to Adaptive Server datatypes.

Table 13. 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

Lists class-level translations from Oracle datatypes to DB2 datatypes.

Table 14. 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 HANA DB

Lists Oracle-to-HANA DB datatype mappings.

See *SAP HANA Reference > SQL Reference Manual > Data Types* for current information on the range of values that can be represented by HANA DB datatypes.

Note: Do not use a column with the Oracle RAW or LONG RAW datatype as a primary key or as part of a primary key.

Table 15. Oracle-to-HANA DB Datatype Mappings

Oracle Datatype	Replication Definition Datatype	HANA DB Datatype
ANYDATA	opaque	No default
BFILE	image	blob
BINARY_DOUBLE	rs_oracle_decimal	double
BINARY_FLOAT	rs_oracle_float	real
BLOB	image	blob
CHAR	char	char (See note below.)
CLOB	text unitext (multibyte)	clob nclob (multibyte)
DATE	rs_oracle_date-time	timestamp
FLOAT	rs_oracle_float	float
INTEGER	rs_oracle_decimal	decimal (to 34 digits) varchar (over 34 digits)
INTERVAL	rs_oracle_inter-val	No default
LONG	text	clob
LONG RAW	image	blob
NCHAR	unichar	nchar
NCLOB	unitext	nclob
NESTEDTAB	opaque	No default

Oracle Datatype	Replication Definition Datatype	HANA DB Datatype
NUMBER	rs_oracle_decimal	double **Note:** When Oracle NUMBER is specified without precision, the maximum value allowed by the Oracle column exceeds HANA DB decimal. HANA DB double is the only HANA DB datatype supporting any possible Oracle value. When the NUMBER precision is known or specified, use HANA DB decimal.
NUMBER(1)	rs_oracle_decimal	tinyint
NUMBER(2)– NUMBER(4)	rs_oracle_decimal	smallint
NUMBER(5)– NUMBER(9)	rs_oracle_decimal	integer
NUMBER(10)– NUMBER(18)	rs_oracle_decimal	bigint
NUMBER(p,s)	rs_oracle_decimal varchar	decimal (to 34 digits) varchar (over 34 digits)
NVARCHAR	univarchar	nvarchar
NVARCHAR2	univarchar	nvarchar
Object types	opaque	No default
RAW	rs_oracle_binary (up to 255 bytes) varbinary (greater than 255 bytes)	varbinary
REF	rs_oracle_binary	No default
ROWID	rs_oracle_rowid	varchar(20)
TIMESTAMP(n)	rs_oracle_timestamp9	timestamp

Oracle Datatype	Replication Definition Datatype	HANA DB Datatype
TIMESTAMP WITH LOCAL TIME ZONE	rs_oracle_time-stamp9	No default
TIMESTAMP WITH TIME ZONE	rs_oracle_time-stamptz	No default
VARCHAR2	varchar	varchar (See note below.)
VARRAY	opaque	No default

Note: If your primary database is configured for a single-byte character set (like iso_1) that accepts extended ASCII characters—the one-byte ASCII character values between 128 and 255—HANA DB may convert single-byte extended characters into double-byte characters in `char` or `varchar` columns. This conversion may subsequently cause the storage size in HANA DB to exceed the original column length or storage of the primary database. If your primary database and application support using extended ASCII characters in a single-byte character set, you may need to increase the size of the corresponding column in the HANA DB table definition. This change allows those columns to accommodate the conversion of extended characters to two-byte storage.

Fixed-Length Column Mapping

HANA DB does not pad fixed-length `CHAR` column values specified in a subscription **where** clause, and this can result in a data mismatch. For example, if the character string 'abc' is passed in a fixed-length column defined in a subscription **where** clause as char(5), HANA DB does not pad the remaining characters with whitespace as 'abc ', and a mismatch occurs. You should therefore map fixed-length `CHAR` columns to the HANA DB `varchar` datatype.

Oracle to Microsoft SQL Server datatypes

Lists class-level translations from Oracle datatypes to Microsoft SQL Server datatypes.

Table 16. 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

Lists the Replication Server user-defined datatype (UDD) names that identify Oracle datatypes.

Table 17. 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*

HANA DB Datatypes

Learn about the class-level translations (default datatype mapping) for HANA DB datatypes and Replication Server datatype names for HANA DB datatypes.

Adaptive Server to HANA DB Datatypes

Lists Adaptive Server-to-HANA DB datatype mappings.

See *SAP HANA Reference > SQL Reference Manual > Data Types* for current information on the range of values that can be represented by HANA DB datatypes.

Table 18. Adaptive Server-to-HANA DB Datatype Mappings

Adaptive Server Datatype	Replication Definition Datatype	HANA DB Datatype
bigdatetime	bigdatetime	timestamp
bigint	bigint	bigint

Adaptive Server Datatype	Replication Definition Datatype	HANA DB Datatype
bigtime	bigtime	No default
binary	binary	varbinary
bit	tinyint	tinyint
char	char	char (See note below.)
date	date	date
datetime	datetime	timestamp
decimal	decimal varchar	decimal (to 34 digits) varchar (over 34 digits)
double precision	float	double
float	float	float
identity	numeric	numeric
int	integer	integer
image	image	blob
longsysname varchar(255) Not Null	varchar(255)	varchar(255)
money	decimal	decimal(19,4)
nchar	char	nchar
numeric	numeric	decimal
nvarchar	varchar	nvarchar
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	decimal	decimal(10,4)
sysname varchar(30) Not Null	varchar(30)	varchar(30)
text	text	clob

Adaptive Server Datatype	Replication Definition Datatype	HANA DB Datatype
timestamp	timestamp	No default
tinyint	tinyint	tinyint
time	time	time
unichar	unichar	nchar
univarchar	univarchar	nvarchar
unsigned bigint		decimal
unsigned int	unsigned int	bigint
unsigned smallint	unsigned smallint	integer
unsigned bigint	unsigned bigint	decimal
unitext	unitext	nclob
varbinary	varbinary	varbinary
varchar	varchar	varchar (See note below.)

Note: If your primary database is configured for a single-byte character set (like iso_1) that accepts extended ASCII characters—the one-byte ASCII character values between 128 and 255—HANA DB may convert single-byte extended characters into double-byte characters in `char` or `varchar` columns. This conversion may subsequently cause the storage size in HANA DB to exceed the original column length or storage of the primary database. If your primary database and application support using extended ASCII characters in a single-byte character set, you may need to increase the size of the corresponding column in the HANA DB table definition. This change allows those columns to accommodate the conversion of extended characters to two-byte storage.

Materialization

Learn about the subscription materialization issues that you must consider when implementing a replication system with heterogeneous or non-ASE data servers, as well as how to materialize subscriptions to primary tables in a non-ASE database.

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.

The types include:

- 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.

See the *Replication Server Administration Guide Volume 1 > Manage Subscriptions* for information about subscription materialization methods.

Heterogeneous Materialization

You may use a bulk materialization or automatic materialization, if it applies to materialize subscriptions to primary data in a non-ASE data server.

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.

The options include:

- 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* > *Replication Server System Functions* > ***rs_marker*** for information about applying this 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.

Unload Data from a Primary Database

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 also

- *Datatype Translation* on page 251

Datatype Translation

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.

Load 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.

See the *Adaptive Server Enterprise Utility Guide > Utility Commands Reference > **bcp***.

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.

Preparation for Materialization

Before you start an atomic bulk materialization, there are things that you need to verify.

You need to verify:

- 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

Learn to perform 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.
   - *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.

See the *Replication Server Reference Manual > Replication Server Commands* and the *Replication Server Administration Guide Volume 1* for information on configuring Replication Servers and materialization methods.

See also

- *IBM DB2 for Linux, UNIX, and Windows as Replicate Data Server* on page 81

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.

Preparation For Materialization

Before you start a nonatomic bulk materialization procedure, there are things that you need to verify.

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

Learn to perform 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.

See also:

- *Replication Server Reference Manual > Replication Server Commands* for information on Replication Command Language (RCL) commands
- *Replication Server Administration Guide Volume 1* for information on configuring Replication Servers and materialization methods

See also

- *IBM DB2 for Linux, UNIX, and Windows as Replicate Data Server* on page 81

Autocorrection

Replication Server can set **autocorrection** on a replication definition to prevent failures that may 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 > > Replication Agent for Microsoft SQL > Replication Server* ***set autocorrection*** *Command* and the *Replication Agent 15.5 Reference Manual > Command Reference >* ***ra_set_autocorrection***.

Direct Load Materialization

Use direct load materialization to materialize data between different kinds of primary and replicate databases.

Direct load materialization differs from other automatic materialization methods:

- No materialization queue is used with direct load materialization. Data is loaded directly from a primary table into a replicate table.
- Replication to other tables is not suspended during direct load materialization. DML operations on a primary table being materialized are stored in a catch-up queue and applied to the replicate table after the initial materialization phase. DML operations on a primary table that is not being materialized are replicated into the replicate table as the DSI receives them. Multiple tables can be concurrently materialized with direct load materialization.
- When subscription materialization stops due to an error, regular replication to other tables is not suspended.
- Multiple parallel threads can be used to load data from one primary table to its corresponding replicate table. You can tune this multi-threaded behavior with **max_mat_load_threads**.

The atomic and nonatomic materialization methods described here are only supported for an Adaptive Server primary. For an explanation of the different types and methods of materialization, see the *Replication Server Heterogeneous Replication Guide > Materialization > Types of Materialization* and the *Replication Server Administration Guide: Volume 1 > Manage Subscriptions > Subscription Materialization Methods*.

Direct load materialization can be used to materialize data:

- from Adaptive Server to HANA DB
- from Microsoft SQL Server to HANA DB
- from Oracle to HANA DB
- from DB2 UDB to HANA DB

Note: Direct load materialization is not supported for materializing data into an Adaptive Server database.

*Restrictions and Limitations for **create subscription***

- When the **direct_load** option is used, no other subscription can be created or defined at the same time for the same replicate table.
- The **direct_load** option is for subscriptions to table replication definitions only and is used with **without holdlock**. It cannot be used with **without materialization** or **incrementally**.
- The **user** and **password** options are used only with **direct_load**.
- You can only use the **direct_load** option against a physical database connection, not an alternate or logical connection. This is the case for both the primary connection—the connection specified in the replication definition—and the replicate connection—the connection specified in the subscription.
- The maintenance user of the primary database cannot be used in the **user** and **password** options to create subscriptions.
- You cannot use atomic materialization if the primary database is not Adaptive Server. For a primary database other than Adaptive Server, the only automatic materialization option supported is direct load.You cannot drop a subscription with the **with purge** option if the replicate database is not Adaptive Server.
- The **direct_load** option is available only if the replicate Replication Server site version and route version are 1571100 or later.
- You can use row filtering, name mapping, customized function strings and datatype mapping with subscriptions created using the **direct_load** option.
- Replication Server rejects any attempt to create a subscription with the **direct_load** option if the number of subscriptions being created has reached or exceeded **num_concurrent_subs**.

Subscriptions and Direct Load Materialization

Direct load materialization is only for use with subscriptions to table replication definitions.

To use direct load materialization, create your subscription to a table replication definition using the **direct_load** option of the **create subscription** command. See *Replication Server Reference Manual > Replication Server Commands > create subscription*.

When you drop a subscription that has been created with the **direct_load** option:

- If the subscription is not yet valid, use the **drop subscription** command with the **without purge** option, which immediately stops the materialization threads, skips any pending DML operations on the table, and drops the subscription.
- If the subscription is not yet valid when it is dropped, the replicate Replication Server DSI must be running for **drop subscription** to complete because Replication Server needs to clear all commands related to the subscription being dropped.

When you drop a subscription with the **without purge** option, it is your responsibility to clean up the replicate table.

Direct Load Materialization from Adaptive Server to HANA DB

Use this procedure to set up direct load materialization and enable replication from an Adaptive Server primary to a HANA DB replicate.

1. Use **isql** to log in to the replicate Replication Server:

   ```
   isql -Uiuser -Pipassword -SRRS_servername
   ```

   where:
   - *iuser* is the subscription creator, who must have **create object** permission.
   - *ipassword* is the password for *iuser*.
   - *RRS_servername* is the server name of the replicate Replication Server.
2. At the replicate Replication Server, create the subscription:

   ```
   1> create subscription subscription_name
   2> for replication_definition
   3> with replicate at dataserver.database
   4> [where search_conditions]
   5> without holdlock
   6> direct_load

   8> [user cusername password cpass]
   9> go
   ```

   The *dataserver.database* must match the Replication Server connection name you use for the replicate database. If you specify *cusername* here, this name is used to log in to the primary database and select from the primary table. If you do not specify *cusername* here, *iuser*—the name specified in **isql**—is used. Make sure that the name used to log in to the primary database has necessary permission at the primary database.
3. Monitor the progress of the subscription materialization:

   ```
   1> check subscription subscription_name for repdef_name
   2> with replicate at replicate_dataserver.replicate_database
   3> go
   ```

   When materialization is complete, Replication Server returns a message similar to:

   ```
   Subscription <subscription_name> has been MATERIALIZED at the
   replicate.
   ```

   When the subscription process is complete, Replication Server returns a message similar to:

   ```
   Subscription <subscription_name> is VALID at the replicate.
   ```

   If there is an error in the subscription process, the **check subscription** command returns a message similar to:

   ```
   Subscription <subscription_name> encountered ERROR.
   ```

   When a subscription encounters an error, look at the replicate and primary Replication Server log files to diagnose the cause, drop the subscription without purge, delete data

from the replicate table, correct the error, and then recreate the subscription. See the *Replication Server Troubleshooting Guide > Subscription Problems > check subscription > Materialization Status* and the *Replication Server Troubleshooting Guide > Subscription Problems > Materialization Problems*.

See the *Replication Server Reference Manual > Replication Server Commands* and the *Replication Server Administration Guide Volume 1* for information on configuring Replication Servers and materialization methods.

Direct Load Materialization from a Non-Adaptive Server Database to HANA DB

Use this procedure to set up direct load materialization and materialize data from a Microsoft SQL Server, Oracle, or DB2 UDB primary to a HANA DB replicate.

Replication Agent is used in direct load materialization from a Microsoft SQL Server, Oracle, or DB2 UDB primary: Replication Server logs in to Replication Agent, and Replication Agent issues queries on behalf of Replication Server, returning query results to Replication Server.

1. Use **dsedit** to update the Replication Server `sql.ini` (Windows) or `interfaces` (UNIX or Linux) file to include an entry for the Replication Agent location.
2. At the Replication Server that manages the primary database, create a connection to the primary database:

   ```
   create connection to pds.pdb
   using profile profile_name;standard
   set username muser
   set password mpwd
   with log transfer on,
   dsi_suspended
   go
   ```

   where:
   - *pds* is the value of the **rs_source_ds** parameter specified in Replication Agent.
   - *pdb* is the value of **rs_source_db** specified in Replication Agent.
   - *profile_name* is **rs_rs_to_msss_ra** for Microsoft SQL Server, **rs_rs_to_oracle_ra** for Oracle, or **rs_rs_to_udb_ra** for DB2 UDB
   - *muser* is the maintenance user for the primary database.
   - *mpwd* is the maintenance user password.

   You only need to do this once.

   If you have an existing primary database connection that does not use one of these connection profiles (**rs_rs_to_msss_ra**, **rs_rs_to_oracle_ra**, or **rs_rs_to_udb_ra**), you must alter that connection so that it uses the correct function string class (**rs_msss_ra_function_class**, **rs_oracle_ra_function_class**, or **rs_udb_ra_function_class**) and the **ra_ra_error_class** error class.
3. Use **isql** to log in to the replicate Replication Server:

   ```
   isql -Uiuser -Pipassword -SRRS_servername
   ```

where:

- *iuser* is the subscription creator, who must have **create object** permission.
- *ipassword* is the password for *user*.
- *RRS_servername* is the server name of the replicate Replication Server.

4. (Optional) Open an **isql** session to Replication Agent with the user ID and password from the following step. This is to ensure that the user ID in the following step has the proper Replication Agent permissions and the correct password.
5. At the replicate Replication Server, create the subscription:

```
1> create subscription subscription_name
2> for replication_definition
3> with replicate at dataserver.database
4> [where search_conditions]
5> without holdlock
6> direct_load

8> [user cusername password cpass]
9> go
```

The *dataserver.database* must match the Replication Server connection name you use for the replicate database.

If you specify *cusername* here, this user is used to connect to Replication Agent. It must be the Replication Agent admin user. If you do not specify *cusername* here, *iuser*—the name specified in **isql**—is used. Make sure that this user is the Replication Agent admin user.

6. Monitor the progress of the subscription materialization:

```
1> check subscription subscription_name for repdef_name
2> with replicate at replicate_dataserver.replicate_database
3> go
```

When materialization is complete, Replication Server returns a message similar to:

```
Subscription <subscription_name> has been MATERIALIZED at the
replicate.
```

When the subscription process is complete, Replication Server returns a message similar to:

```
Subscription <subscription_name> is VALID at the replicate.
```

If there is an error in the subscription process, the **check subscription** command returns a message similar to:

```
Subscription <subscription_name> encountered ERROR.
```

When a subscription encounters an error, look at the replicate and primary Replication Server log files to diagnose the cause, drop the subscription without purge, delete data from the replicate table, correct the error, and then recreate the subscription. See the *Replication Server Troubleshooting Guide > Subscription Problems > check subscription > Materialization Status* and the *Replication Server Troubleshooting Guide > Subscription Problems > Materialization Problems*.

When you complete this procedure, the subscription is created, the replicate data is consistent with the primary data, and replication of this table is enabled.

See the *Replication Server Reference Manual > Replication Server Commands* and the *Replication Server Administration Guide Volume 1* for information on configuring Replication Servers and materialization methods.

Direct Load Materialization Configuration Parameters

Use these configuration parameters with direct load materialization.

rs_init sets default configuration parameters after you install your Replication Server.

See *Replication Server Administration Guide Volume 1 > Manage a Replication System > Set Replication Server Configuration Parameters > Change Replication Server Parameters* for information on how to modify these parameters using **configure replication server**.

Table 19. Direct Load Materialization Configuration Parameters

Configuration parameter	Description
mat_load_tran_size	Specifies the optimal transaction size or batch size for the initial copying of primary data to the replicate table during direct load materialization. Valid values: 10 – 2147483646 Default: 10000
max_mat_load_threads	Specifies the maximum number of load threads for each table being materialized. Valid values: 1 – 20 Default: 5 Replication Server begins direct load materialization with one load thread for each table and spawns more threads as necessary up to the number specified by this parameter. **max_mat_load_threads** is a local Replication Server and database connection parameter. The value of this parameter and **num_concurrent_subs** control resource use for direct load materialization.

Heterogeneous Database Reconciliation

Learn about the issues involved with comparing and reconciling data from different databases in a heterogeneous replication system.

Sybase rs_subcmp Utility

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.

Replication Server Data Assurance Option

Replication Server Data Assurance Option compares row data and schema between two or more databases, and reports discrepancies.

Replication Server Data Assurance Option is a scalable, high-volume, and configurable data comparison product, allowing you to run comparison jobs even during replication by using a "wait and retry" strategy that eliminates any down time.

Each comparison job lets you check for data discrepancies using a number of settings that determine which data is being compared and in what way. Replication Server Data Assurance Option includes a command line tool (CLT) that allows users to perform all comparison and reporting jobs. Users can monitor and abort jobs, as well as generate detailed comparison reports.

Replication Server Data Assurance Option allows large tables to be split into multiple partitions for comparing data in parallel. You can also compare row data between any combination of Adaptive Server® and Sybase® IQ, Oracle, or SAP® HANA® databases in a heterogeneous comparison environment.

Replication Server Data Assurance Option is licensed through SySAM license manager and is available on multiple platforms. For more information about SySAM, see the installation guide for your platform, or the SySAM Web site: *http://www.sybase.com/sysam*.

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.

See the *Replication Server Administration Guide* and the *Replication Server Reference Manual > Executable Programs >* ***rs_subcmp*** for more information on **rs_subcmp**.

Troubleshoot Heterogeneous Replication Systems

Learn to troubleshoot the common problems in Sybase replication systems with heterogeneous or non-ASE data servers.

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.

Inbound 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).

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

Learn to determine the reason for unupdated inbound queue.

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

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).

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.

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.

Determining the Reason the Outbound Queue Is Not Being Updated

Learn to determine the reason for unupdated outbound queue.

1. Verify that any Replication Server routes are active.

   See the *Replication Server Troubleshooting Guide > Route Problems* 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.

   See the *Replication Server Administration Guide* for information about recovering from this error.
4. Verify the primary replication definition.

Determining Why Replicate Database Is Not Updated

Learn to determine why replicate transactions are not applied at the replicate database.

If the Replication Server outbound queue is being updated but transaction data is not being applied at the replicate database, use this procedure to determine the reason:

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.
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.
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. See the *Replication Server Troubleshooting Guide* for more information about Replication Server trace flags.
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**

   See the appropriate ECDA documentation for specific trace availability and syntax.

See also

- *Expected Datatype Translations Do Not Occur* on page 272
- *Updates to rs_lastcommit Fail* on page 272

HDS Issues and Limitations

Learn about some of the 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 must be stopped.

This must be 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
   '9999999999999999999.9999999999999999999' 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>'
```

Troubleshoot Specific Errors

Learn about how to troubleshoot specific errors that 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.

Troubleshooting rs_lastcommit Update Failure

Identify the problem if the replicate Replication Server error log displays a syntax error while updating the `rs_lastcommit` table.

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.

   If a replicate database connection uses ExpressConnect for Oracle or ExpressConnect for HANA DB, turn on tracing in ExpressConnect.

   ```
   alter connection to <rds.rdb> set trace to 'econn,dsi_buf_dump,on'
   go
   ```

   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

Learn to validate 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**
   - HANA DB – **rs_hanadb_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.

   See the *Replication Server Administration Guide Volume 1 > Manage Replicated Tables > Translate Datatypes Using HDS* for more information about heterogeneous datatype support (HDS) and the **admin translate** command.

Missing Column Values

Missing column values may result from several Replication Server configurations.

Some column values may be missing at the replicate table if:

- Row migration occurs in a subscription that has a **where**. See *Row Migration* in the *Replication Server Administration Guide Volume 1*.
- You enable autocorrection for each replication definition. See *Nonatomic Materialization* and *Autocorrection for Nonatomic Materialization* in the *Replication Server Administration Guide Volume 1*.
- You set **dsi_command_convert** to **u2di** to convert commands. See *Command Conversion* in the *Replication Server Administration Guide Volume 2*.
- You customize the **rs_update** system function to execute an **insert** operation. See *Customize Database Operations* in the *Replication Server Administration Guide Volume 2*.

To prevent missing column values, enable the **ra_set_autocorrection** Replication Agent command to allow Replication Agent to send the values for all columns to the downstream Replication Server:

```
ra_set_autocorrection tablename, enable
```

Note: Enabling **ra_set_autocorrection** can impact replication performance and may not prevent missing LOB values in columns in the replicate database..

See **ra_set_autocorrection** in the *Replication Agent Reference Manual*.

Log Transfer Language Generation and Tracing

Learn about the information on 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 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 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, see the *Replication Agent Administration Guide*.

Reference Implementation for Oracle to Oracle Replication

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.

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 Oracle Reference Implementation

You must have supported versions of the components of a replication environment before you can implement a reference environment.

For Oracle reference implementation, these product component versions are supported:

Table 20. Supported Product Component Versions for Oracle Reference Implementation

Replication Server	Oracle	Replication Agent for Oracle	ExpressConnect for Oracle
15.5, 15.6, 15,7, 15.7.1	11g Release 2	15.7.1 SP100	15.7.1 SP100

For example, you can build a reference implementation environment for Oracle with Replication Server 15.7.1, Oracle 11g Release 2, Replication Agent 15.7.1 SP100 for Oracle, and ExpressConnect for Oracle 15.7.1 SP100.

Prerequisites for the Reference Environment

There are several prerequisites and some information you must be aware of before you build the reference environment.

1. For Oracle, verify that you have **execute** permission in the Oracle release directory. For example, verify whether you can manually create an instance.
2. Verify that the environment variable settings in the `SYBASE.sh` file in the Replication Server or Adaptive Server release directory is correct. If you cannot verify this, remove or rename the file.
3. Verify that you have the UNIX **grep**, **kill**, **awk**, and **ps** commands available in your bash shell.

The reference implementation procedure uses the `interfaces` file in the Replication Server release directory. If the file exists before you run the reference implementation procedure, the procedure backs up the existing file by incrementing the file name extension.

For Oracle, the reference implementation procedure renames the existing `tnsname.ora`, `listener.ora` files and creates new files for the Oracle reference implementation.

Reference Implementation Configuration Files for Oracle

Sybase provides configuration file templates for Oracle-to-Oracle replication on supported UNIX and Microsoft Windows platforms, that you can use to create a configuration file for your environment.

The files are located in `$SYBASE/REP-15_5/REFIMP-01_0`.

Table 21. Reference Implementation Configuration Files

Primary to replicate data server and platform	Configuration file
Oracle-to-Oracle on UNIX	`ora_unix_refimp.cfg`
Oracle-to-Oracle on Windows	`ora_win_refimp.cfg`

See *Replication Server Administration Guide Volume 2 > Implement a Reference Replication Environment* for requirements, instructions, a sample configuration file, and the objects created by implementing the reference environment.

Glossary

Glossary of terms used in replication systems.

- **active database –** In a warm standby application, a database that is replicated to a standby database. See also *warm standby application*.
- **Adaptive Server –** The Sybase version 11.5 and later relational database server. If you choose the RSSD option when configuring Replication Server, Adaptive Server maintains Replication Server system tables in the RSSD database.
- **application programming interface (API) –** A predefined interface through which users or programs communicate with each other. Open Client and Open Server are examples of APIs that communicate in a client/server architecture. RCL, the Replication Command Language, is the Replication Server API.
- **applied function –** A replicated function, associated with a function replication definition, that Replication Server delivers from a primary database to a subscribing replicate database. The function passes parameter values to a stored procedure that is executed at the replicate database. The stored procedure executed at the replicate database by the maintenance user. See also *replicated function delivery*, *request function*, and *function replication definition*.
- **article –** A replication definition extension for tables or stored procedures that can be an element of a publication. Articles may or may not contain **where** clauses, which specify a subset of rows that the replicate database receives.
- **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 where 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 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 –** Autocorrection is 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. Autocorrection should

only be enabled for replication definitions whose subscriptions use nonatomic 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 matching the subscription's bitmap are replicated.
- **bulk copy-in –** A feature that improves Replication Server performance when replicating large batches of **insert** statements on the same table in Adaptive Server® Enterprise 12.0 and later. Replication Server implements bulk copy-in in Data Server Interface (DSI), the Replication Server module responsible for sending transactions to replicate databases, using the Open Client™ Open Server™ Bulk-Library.

  Bulk copy-in also improves the performance of subscription materialization. When **dsi_bulk_copy** is on, Replication Server uses bulk copy-in to materialize the subscriptions if the number of **insert** commands in each transaction exceeds **dsi_bulk_threshold**.
- **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, diskette, CD-ROM, 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*.
- **centralized database system –** A database system where data is managed by a single database management system at a centralized location.
- **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. It 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.
- **concurrency –** The ability of multiple clients to share data or resources. Concurrency in a database management system depends upon the system protecting clients from conflicts that arise when data in use by one client is modified by another client.
- **connection –** A connection from a Replication Server to a database. See also *Data Server Interface (DSI)* and *logical connection*.
- **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 generation number –** Stored in both the database and the RSSD of the Replication Server that manages the database, the database generation number is the first part of the origin queue ID (*qid*) of each log record. The origin queue ID ensures that the Replication Server does not process duplicate records. During recovery operations, you may need to increment the database generation number so that Replication Server does not ignore records submitted after the database is reloaded.
- **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, 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 corresponding 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 communications method used to access that DBMS from other parts of a replication system. See also *database* and *data server*.
- **decision support application –** A database client application characterized by ad hoc queries, reports, and calculations and few data update transactions.

- **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 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*.
- **direct route –** A route used to send messages directly from a source to a destination Replication Server, with no intermediate Replication Servers. See also *indirect route* and *route*.
- **disk partition –** See *partition*.
- **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 that Replication Server use the dump marker to determine where in the transaction stream to begin applying transactions in the standby database. See also *warm standby application*.
- **Embedded Replication Server System Database (ERSSD) –** The SQL Anywhere (SA) database that stores Replication Server system tables. You can choose whether to store Replication Server system tables on the ERSSD or the Adaptive Server RSSD. See also *Replication Server System Database (RSSD)*.
- **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.
- **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.
- **exceptions log –** A set of three Replication Server system tables that holds information about transactions that failed on a data server. The transactions in the log must be resolved by a user or by an intelligent application. You can use the **rs_helpexception** stored procedure to query the exceptions log.
- **ExpressConnect for HANA DB –** A set of libraries that can be used to provides direct communication between Replication Server and a HANA DB database.
- **ExpressConnect for Oracle –** A set of libraries that can be used to provides direct communication between Replication Server and an Oracle database.
- **Failover –** Sybase Failover allows you to configure two version 12.0 and later Adaptive Servers as companions. If the primary companion fails, that server's devices, databases, and connections can be taken over by the secondary companion.

  For more detailed information about how Sybase Failover works in Adaptive Server, refer to *Using Sybase Failover in a High Availability System*, which is part of the Adaptive Server Enterprise documentation set.
- **fault tolerance –** The ability of a system to continue to operate correctly even though one or more of its component parts is malfunctioning.
- **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. In order 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 function*, 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. There are two types of function replication definition, applied and request. 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. The three system-provided function-

string classes are `rs_sqlserver_function_class`, `rs_default_function_class`, and `rs_db2_function_class`. 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 run time. 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 both applied and request function delivery).
- **gateway –** Connectivity software that allows two or more computer systems with different network architectures to communicate.
- **generation number –** See *database generation number*.
- **heterogeneous data servers –** Multi-vendor data servers used together in a distributed database system.
- **hibernation mode –** A Replication Server state in which all DDL commands, except **admin** and **sysadmin** commands, are rejected; all routes and connections are suspended; most service threads, such as DSI and RSI, are suspended; and RSI and RepAgent users are logged off and not allowed to log on. Used during route upgrades, and may be turned on for a Replication Server to debug problems.
- **high-performance analytic appliance (HANA) –** An SAP® in-memory online transaction processing and online analytical processings solution.
- **high-performance analytic appliance database (HANA DB) –** The SAP in-memory database.
- **high availability (HA) –** Very low downtime. Computer systems that provide HA usually provide 99.999% availability, or roughly five minutes unscheduled downtime per year.
- **high volume adaptive replication (HVAR) –** Compilation of a group of **insert**, **delete**, and **update** operations to produce a net result and the subsequent bulk application of the net result to the replicate database.
- **hot standby application –** A database application in which the standby database can be placed into service without interrupting client applications and without losing any transactions. See also *warm standby application*.
- **ID Server –** One Replication Server in a replication system is the ID Server. In addition to performing the usual Replication Server tasks, the ID Server assigns unique ID numbers to every Replication Server and database in the replication system, and maintains version information for the replication system.
- **inbound queue –** A stable queue used to spool messages from a Replication Agent to a Replication Server.

- **indirect route –** A route used to send messages from a source to a destination Replication Server, through one or more intermediate Replication Servers. See also *direct route* and *route*.
- **interfaces 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, gateways, Replication Servers, and Replication Agents. 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.
- **local-area network (LAN) –** A system of computers and devices, such as printers and terminals, connected by cabling for the purpose of sharing data and devices.
- **locator value –** The value stored in the `rs_locater` table of the Replication Server's RSSD that identifies the latest log transaction record received and acknowledged by the Replication Server from each previous site during replication.
- **logical connection –** A database connection that Replication Server maps to the connections for the active and standby databases in a warm standby application. See also *connection* and *warm standby application*.
- **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 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 *RepAgent 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 –** The process of 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.
- **mixed-version system –** A replication system containing Replication Servers of different software versions that have different capabilities based on their different software versions and site versions. Mixed-version support is available only if the system version is 11.0.2 or greater.

  For example, a replication system containing Replication Servers version 11.5 or later and version 11.0.2 is a mixed-version system. A replication system containing Replication Servers of releases earlier than release 11.0.2 is not a mixed-version system, because any

newer Replication Servers are restricted by the system version from using certain new features. See also *site version* and *system version*.

- **more columns –** Columns in a replication definition exceeding 250, but limited to 1024. More columns are supported by Replication Server version 12.5 and later.
- **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*.
- **Multi-Path Replication™ –** Replication Server feature that improves performance by enabling parallel paths of data from the source database to the target database,You can configure multi-path replication in warm standby and multisite availability (MSA) environments. These multiple paths process data independently of each other and are applicable when sets of data can be processed in parallel without transactions consistency requirements between them. while still maintaining data consistency within a path, but not adhering to the commit order across different paths.
- **name space –** 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. You can use this 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 RepAgent, the `qid` uniquely identifies each log record passed to the Replication Server. It includes the `date` and `timestamp` and the database generation number. See also *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 –** 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*.
- **partition –** A raw disk partition or operating system file that Replication Server uses for stable queue storage. Only use operating system files in a test environment.
- **physical connection –** See *connection*.
- **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 via the replication system.
- **primary fragment –** A horizontal segment of a table that holds the primary version of a set of rows.
- **primary key –** A set of table columns that uniquely identifies each row.
- **primary site –** A Replication Server where a function-string class or error class is defined. See *error class* and *function-string class*.
- **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.
- **projection –** A vertical slice of a table, representing a subset of the table's columns.
- **publication –** A group of articles from the same primary database. A publication lets you collect replication definitions for related tables and/or stored procedures and then subscribe to them as a group. You 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. See also *article* and *publication subscription*.
- **publication subscription –** A subscription to a publication. See also *article* and *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 query is a request to retrieve data that meets a given set of criteria. The SQL database language includes the **select** command for queries.

- **quiescent –** A quiescent replication system is one in which all updates have been propagated to their destinations. Some Replication Server commands or procedures require that you first quiesce the replication system.
- **quoted identifiers –** Object names that contain special characters such as spaces and non-alphanumeric characters, start with a character other than an alphabet, or that correspond to a reserved word, need to be enclosed in double quote characters to be parsed correctly.
- **real time loading (RTL) –** High volume adaptive replication (HVAR) to a Sybase IQ database. Uses relevant commands and processes to apply HVAR changes to a Sybase IQ replicate database. See *high volume adaptive replication*.
- **remote procedure call (RPC) –** A request to execute a procedure that resides in a remote server. The server that executes the procedure could be a 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.
- **RepAgent thread –** The Replication Agent for Adaptive Server databases. RepAgent is an Adaptive Server thread; it transfers transaction log information from the primary database to a Replication Server for distribution to other databases.
- **replicate database –** Any database that contains data that is replicated from another database via 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 transfers transaction log information representing modifications made to primary data from a database server to a Replication Server for distribution to other databases. RepAgent is the Replication Agent for Adaptive Server databases.
- **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* and *route*.
- **Replication Monitoring Services (RMS) –** A small Java application built using the Sybase Unified Agent Framework (UAF) that monitors and troubleshoot a replication environment.
- **replication system administrator –** The system administrator that 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 SQL Anywhere (SA) ERSSD. See also *Embedded Replication Server System Database (ERSSD)*.
- **Replication Server system Adaptive Server –** The Adaptive Server with the database containing a Replication Server's 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 upon 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 primary database to a replicate database. The function passes parameter values to a stored procedure that is executed at the replicate database. The stored procedure is executed at the replicate site by the same user as it is at the primary site.See also *replicated function delivery*, *request function*, and *function replication definition*.
- **resync marker –** When you restart Replication Agent in resync mode, Replication Agent sends the resync database marker to Replication Server to indicate that a resynchronization effort is in progress. The resync marker is the first message Replication Agent sends before sending any SQL data definition language (DDL) or data manipulation language (DML) transactions.
- **route –** A one-way message stream from a source Replication Server to a destination Replication Server. Routes carry data modification commands (including those for RSSDs) and replicated functions or stored procedures between Replication Servers. See also *direct route* and *indirect route*.

- **route version –** The lower of the site version numbers of the route's source and destination Replication Servers. Replication Server version 11.5 and later use the route version number to determine which data to send to the replicate site. See also *site version*.
- **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.
- **SQL statement replication –** In SQL statement replication, the Replication Server receives the SQL statement that modified the primary data, rather than the individual row changes from the transaction log. Replication Server applies the SQL statement to the replicated site. RepAgent sends both the SQL Data Manipulation Language (DML) and individual row changes. Depending on your configuration, Replication Server chooses either individual row change log replication or SQL statement replication.
- **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. See also *primary site*.
- **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*, *route 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 *inbound queue*, *outbound queue*, and *materialization queue*.

- **standalone mode –** A special Replication Server mode used for initiating recovery operations.
- **standby database –** In a warm standby application, a database that receives data modifications from the active database and serves as a backup of that database. See also *warm standby application*.
- **stored procedure –** A collection of SQL statements and optional control-of-flow statements stored under a name in a 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 Sybase and Powersoft products. Replication Server uses Replication Manager as a Sybase Central plug-in. See also *Replication Monitoring Services (RMS)*.
- **symmetric multiprocessing (SMP) –** On a multiprocessor platform, the ability of an application's threads to run in parallel. Replication Server supports SMP, which can improve server performance and efficiency.
- **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 –** Replication Server provides the error class `rs_sqlserver_error_class` and the function-string classes `rs_sqlserver_function_class`, `rs_default_function_class`, and `rs_db2_function_class`. 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 release 11.0.2 or earlier, and below which no Replication Server can be downgraded or installed. For a Replication Server version 11.5, your use of certain new features requires a site version of 1150 and a system version of at least 1102. See also *mixed-version system*, *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. It 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 RepAgent 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 function –** 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 –** *mixed-version system*

  See *mixed-version system*, *site version*, *software version*, and *system version*.
- **warm standby application –** An application that employs Replication Server to maintain a standby database for a database known as the active database. If the active database fails, Replication Server and client applications can switch to the standby database.
- **wide-area network (WAN) –** A system of local-area networks (LANs) connected together with data communication lines.
- **wide columns –** Columns in a replication definition containing `char`, `varchar`, `binary`, `varbinary`, `unichar`, `univarchar`, or Java `inrow` data that are wider that 255 bytes. Wide columns are supported by Replication Server version 12.5 and later.
- **wide data –** Wide data rows, limited to the size of the data page on the data server. Adaptive Server supports page sizes of 2K, 4K, 8K, and 16K. Wide data is supported by Replication Server version 12.5 and later.
- **wide messages –** Messages larger that 16K that span blocks. Wide messages are supported by Replication Server version 12.5 and later.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

Z