SYBASE®

Reference Manual

Replication Server®

15.0.1

DOCUMENT ID: DC32410-01-1501-02

LAST REVISED: September 2007

About This Book

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

This book, *Replication Server Reference Manual*, describes these Replication Server features:

- The Replication Command Language (RCL) used by Replication Server
- System functions for Replication Server
- Adaptive Server® commands and system procedures that you use with Replication Server

  Note Since changing the name of Sybase SQL Server® to Adaptive Server Enterprise, Sybase uses the names Adaptive Server and Adaptive Server Enterprise to refer collectively to all supported versions of Sybase SQL Server and Adaptive Server Enterprise. From this point forward, in this document, Adaptive Server Enterprise is referred to as Adaptive Server.

- Adaptive Server stored procedures that you use to manage the Replication Server system tables
- Replication Server executable programs, which you invoke directly from the operating system
- Replication Server system tables

Audience

The *Replication Server Reference Manual* is intended for anyone who uses Replication Server. It is a reference manual, and it assumes that you have basic knowledge of how to use Replication Server.

This book is also for replication system Administrators, who manage the routine operation of Replication Servers. Any user who has sa permission can be a replication system Administrator, although each Replication Server usually has just one.

How to use this book

The information in this book is organized as follows:

- Chapter 1, "Introduction to the Replication Command Language" categorizes the commands and what they do.
- Chapter 2, "Topics" discusses datatypes, identifiers, reserved words, and support for Adaptive Server.
- Chapter 3, "Replication Server Commands" consists of reference pages for all Replication Server commands.
- Chapter 4, "Replication Server System Functions" provides reference pages for each system function Replication Server propagates from primary to replicate databases.
- Chapter 5, "Adaptive Server Commands and System Procedures" contains reference pages for the Adaptive Server commands and system procedures used with Replication Server.
- Chapter 6, "Adaptive Server Stored Procedures" contains reference pages for the Adaptive Server stored procedures used to manage the Replication Server system tables.
- Chapter 7, "Executable Programs" contains reference pages for the Replication Server executable programs and the rs_subcmp subscription comparison program.
- Chapter 8, "Replication Server System Tables" describes each Replication Server system table.
- Chapter 9, "Replication Monitoring Services API" contains reference pages for the Replication Monitor Service (RMS) API.
- Appendix A, "Acronyms and Abbreviations" lists the acronyms and abbreviations used in the Replication Server documentation and system messages.
- Appendix B, "Replication Server Design Limits" lists the maximum and minimum parameters and values for various replication system objects.
- Appendix C, "RMS Server and Component States" lists the RMS server and component states.
- Appendix D, "Event Trigger Arguments" lists the information RMS passes concerning the execution of a certain event.

Related documents

The Replication Server documentation set consists of the following:

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

A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase® Technical Library.

- *Installation Guide* for your platform – describes installation and upgrade procedures for all Replication Server and related products.
- *What's New in Replication Server?* – describes the new features in Replication Server version 15.0.1 and the system changes added to support those features.
- *Administration Guide* – contains an introduction to replication systems. This manual includes information and guidelines for creating and managing a replication system, setting up security, recovering from system failures, and improving performance.
- *Configuration Guide* for your platform – describes configuration procedures for all Replication Server and related products, and explains how to use the rs_init configuration utility.
- *Design Guide* – contains information about designing a replication system and integrating heterogeneous data servers into a replication system.
- *Getting Started with Replication Server* – provides step-by-step instructions for installing and setting up a simple replication system.
- *Heterogeneous Replication Guide* – describes how to use Replication Server to replicate data between databases supplied by different vendors.
- *Reference Manual* (this book) – contains the syntax and detailed descriptions of Replication Server commands in the Replication Command Language (RCL); Replication Server system functions; Adaptive Server commands, system procedures, and stored procedures used with Replication Server; Replication Server executable programs; and Replication Server system tables.
- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Troubleshooting Guide* – contains information to aid in diagnosing and correcting problems in the replication system.
- Replication Manager plug-in help, which contains information about using Sybase Central™ to manage Replication Server.

Other sources of information

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

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

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

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

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

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

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

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

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

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

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

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

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

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

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

Conventions

This section describes style and syntax conventions, RCL command formatting conventions, and graphic icons used in this book.

Style conventions Syntax statements (displaying the syntax and options for a command) are printed as follows:

alter user *user*
set password *new_passwd*
[verify password *old_passwd*]

See “Syntax conventions” on page xviii for more information.

Examples that show the use of Replication Server commands are printed as follows:

```
alter user louise
 set password somNIfic
 verify password EnnuI
```

Command names, command option names, program names, program flags, keywords, configuration parameters, functions, and stored procedures are printed as follows:

Use alter user to change the password for a login name.

Variables, parameters to functions and stored procedures, and user-supplied words are in italics in syntax and in paragraph text, as follows:

The set password *new_passwd* clause specifies a new password.

Names of database objects, such as databases, tables, columns, and datatypes, are in italics in paragraph text, as follows:

The base_price column in the Items table is a money datatype.

Names of replication objects, such as function-string classes, error classes, replication definitions, and subscriptions, are in italics, as follows:

rs_default_function_class is a default function-string class.

Syntax conventions Syntax formatting conventions are summarized in the following table. Examples combining these elements follow.

Table 1: Syntax formatting conventions

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

Obligatory choices

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

```
{red | yellow | blue}
```

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

```
{cash, check, credit}
```

Optional choices

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

```
[anchovies]
```

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

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

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

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

Repeating elements

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

alter function replication definition *function_rep_def*
{deliver as '*proc_name*' |
 add *@parameter datatype* [, *@parameter*
datatype]... |
 add searchable parameters *@parameter*

```
  [, @parameter]... |
send standby {all | replication definition}
  parameters}
```

RCL command formatting

RCL commands are similar to Transact-SQL® commands. The following sections present the formatting rules.

Command format and command batches

- You can break a line anywhere except in the middle of a keyword or identifier. You can continue a character string on the next line by typing a backslash (\) at the end of the line.
- Extra space characters on a line are ignored, except after a backslash. Do not enter any spaces after a backslash.
- You can enter more than one command in a batch, unless otherwise noted.
- RCL commands are not transactional. Replication Server executes each command in a batch without regard for the completion status of other commands in the batch. Syntax errors in a command prevent Replication Server from parsing subsequent commands in a batch.

Case sensitivity

- Keywords in RCL commands are not case-sensitive. You can enter them with any combination of uppercase or lowercase letters.
- 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.

Identifiers

Identifiers are names you give to servers, databases, variables, parameters, database objects, and replication objects. Database object names include names for tables, columns, and views. Replication object names include names for replication definitions, subscriptions, functions, and publications.

- Identifiers can be 1 – 255 bytes long (equivalent to 1 – 255 single-byte characters) and must begin with a letter, the @ sign, or the _ character. See “Identifiers,” in Chapter 2, “Topics” for a list of identifiers that have been extended to 255 bytes.
- Replication Server function parameters are the only identifiers that can begin with the @ character. Function parameter names can include 255 characters *after* the @ character.

- After the first character, identifiers can include letters, digits, and the #, $, or _ characters. Spaces are not allowed.

Parameters in function strings

- Parameters in function strings have the same rules as identifiers, except:
  - They are enclosed in question marks (?), allowing Replication Server to locate them in the function string. Use two consecutive question marks (??) to represent a literal question mark in a function string.
  - The exclamation point (!) introduces a parameter modifier that indicates the source of the data that will be substituted for a parameter at runtime. For a complete list of modifiers, see create function string on page 202.

Data support Replication Server supports all Adaptive Server datatypes.

User-defined datatypes are not supported. The timestamp, double precision, nchar, and nvarchar datatypes are indirectly supported by mapping them to other datatypes. Columns using the timestamp datatype are mapped to varbinary(8).

For more information about the supported datatypes, including how to format them, see “Datatypes” on page 21.

Replication Server supports a set of datatype definitions for non-Sybase data servers that lets you replicate column values of one datatype to a column of a different datatype in the replicate database. See the *Replication Server Administration Guide Volume 1* for more information about heterogeneous datatype support (HDS).

Icons

Illustrations in this book use icons to represent the components of a replication system.

	Description
⊕	This icon represents Replication Server, the Sybase server program maintains replicated data on a local-area network (LAN) and processes data transactions received from other Replication Servers on wide-area network (WAN).
◐	This icon represents Adaptive Server, the Sybase data server. Data servers manage databases containing primary or replicated data. Replication Server also works with heterogeneous data servers, so, unless otherwise noted, this icon can represent any data server in a replication system.

	Description
	This icon represents Replication Agent, a replication system process or module that transfers transaction log information for primary database to a Replication Server. The Replication Agent for Adaptive Server is RepAgent. Sybase provides Replication Agent products for Adaptive Server Anywhere, DB2, Informix, Microsoft SQL Server, and Oracle data servers. Except for RepAgent, which is an Adaptive Server thread, all Replication Agents are separate processes. In general, this icon only appears when representing a Replication Agent that is a separate process.
	This icon represents client application. A client application is a user process or application connected to a data server. It may be a front-end application program executed by a user or a program that executes as an extension of the system.
	This icon represents the Sybase Central Replication Manager plug-in (RM), a management utility that lets a replication system administrator develop, manage, and monitor a Sybase Replication Server environment.

Accessibility features

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

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

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

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

If you need help

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

CHAPTER 1

Introduction to the Replication Command Language

The Replication Command Language (RCL) is divided into these categories:

Topic	Page
Data replication commands	2
User commands	8
Database interface commands	9
Route commands	13
System information commands	13
Partition commands	15
Configuration commands	16
System administration commands	16
Recovery commands	18

This chapter lists and summarizes the commands in each category. Some commands are included in multiple categories. For complete command syntax and usage information, see Chapter 3, "Replication Server Commands."

For detailed information on datatypes, identifiers, reserved words, and support for Adaptive Server, see Chapter 2, "Topics."

For an introduction to Replication Server architecture, see Chapter 1, "Introduction," and Chapter 2, "Replication Server Technical Overview" in the *Replication Server Administration Guide Volume 1*.

Some Replication Server procedures may require you to execute Adaptive Server system procedures such as sp_setreptable or sp_setrepproc. For complete syntax and usage information, see Chapter 5, "Adaptive Server Commands and System Procedures."

The Replication Manager (RM) provides another way to perform many of the tasks that RCL commands perform. See *Replication Server Administration Guide Volume 1* for more information.

Data replication commands

Data replication commands create and manage the replication definitions, publications, and subscriptions that make it possible to replicate tables or stored procedures.

Table replication definition commands

A table replication definition describes the table and the columns that are to be replicated. A primary table is the replication source; a replicate table is the destination. You can create one or more replication definitions for each primary table.

Create a replication definition in the Replication Server that manages the database where the primary table is stored.

The replication definition includes:

- A name for the replication definition
- The names of the primary and replicate tables, if they are different from each other and from the replication definition name
- The location of the primary table
- The names and datatypes of the primary columns to be replicated and the corresponding replicate column names
- The names of the columns that form the primary key for the table

The replication definition can optionally include:

- The names of columns that can be referenced in where clauses for subscriptions
- Whether the replication definition and its columns will be used in replicating to a standby database
- Whether to replicate all columns or the minimum number of required columns for update and delete operations
- Replication status for text, unitext, image, and rawobject columns
- Whether to change the datatype of replicated values from the datatype of the primary database to the datatype of the replicate database.

No data is distributed when you create a replication definition. You must create a copy of the table in each replicate database and then create subscriptions to begin replicating data.

Use these commands to work with table replication definitions:

- create replication definition – creates a replication definition for a table.
- alter replication definition – changes a replication definition.
- drop replication definition – removes a replication definition.

For commands that you use in subscribing to replication definitions, see "Subscription commands" on page 5.

Function replication definition commands

A function replication definition specifies information about a stored procedure that is to be replicated. The replication source and destination databases can be primary or replicate databases.

Create a function replication definition in the Replication Server that manages the primary database.

The function replication definition includes:

- A name for the function replication definition. This name is usually the same as that of the stored procedure to be executed in the source database. The only exception is @rs_repdef.
- The location of the primary data.
- The names and datatypes of the stored procedure parameters to be replicated.

The function replication definition can optionally include:

- The name of the stored procedure to be executed in the destination database, if it is different from the name of the function replication definition.
- The names of parameters that can be referenced in where clauses for subscriptions.
- Whether the function replication definition and its parameters will be used in replicating to a standby database.

Use these commands to work with function replication definitions:

- create function replication definition – creates a replication definition for a stored procedure.
- alter function replication definition – changes a function replication definition.
- drop function replication definition – removes a function replication definition.

No data is distributed when you create a function replication definition. You must create stored procedures in both the source and destination databases.

If a stored procedure is to be replicated from a primary to a replicate database, you must create a subscription at the replicate Replication Server. You don’t need a subscription for a stored procedure that is to be replicated from a replicate to a primary database.

See “Subscription commands” on page 5 for commands that you use in subscribing to replication definitions.

Publication commands

The publications feature of Replication Server lets you group the tables and procedures you want to subscribe to, and their replication definitions, and create one subscription for the group.

A **publication** is a set of articles from the same primary database. Each **article** is a replication definition for a table or stored procedure and a set of where clauses that specify which rows are of interest. An article can contain zero, one, or multiple where clauses. Multiple clauses are separated by the or keyword.

Use these commands to work with publications and articles:

- create publication – creates a publication.
- drop publication – removes a publication and its articles. The *drop_repdef* option drops the associated replication definitions.
- validate publication – verifies that a publication has at least one article and marks the publication so that new subscriptions can be created for it.
- check publication – indicates whether subscriptions can be created for a publication, and reports the number of articles it contains.
- create article – creates an article and assigns it to a publication.
- drop article – removes an article from a publication. The *drop_repdef* option also drops the associated replication definition.

See "Publication subscription commands" on page 7 for information about the commands used in subscribing to publications.

Subscription commands

Subscriptions initiate the replication of data or stored procedures. A subscription specifies a table or function replication definition name, or a publication, and the database into which the data is to be replicated.

- A subscription for a table replication definition replicates data.
- A subscription for a function replication definition replicates stored procedures.
- A subscription for a publication replicates the data represented by each article in the publication. Publications can also have articles for stored procedures.

A subscription to a table or function replication definition may include a where clause, which determines the rows that are replicated or whether a stored procedure is replicated.

Note A subscription to a publication cannot include a where clause. where clauses are contained in the publication's articles.

Subscription materialization

When you create a subscription for a table replication definition, rows that fit the subscription are copied from the primary to the replicate table in a process called **materialization**. After materialization is complete, Replication Server distributes row changes in the primary database through normal replication.

If a subscription involves many rows, materialization can hold locks for a long time and overload the network. Replication Server queues may also fill with data. To avoid these problems, Replication Server provides four different ways to materialize a subscription.

You can use any method for subscriptions for table replication definitions or for publications. Use nonmaterialization or bulk materialization for subscriptions for function replication definitions.

- **Atomic materialization** is the default method for table replication definitions. Replication Server selects rows at the primary table, using a holdlock, and copies them over the network. The primary table is locked during materialization and data is consistent between the primary and replicate tables.
- In **nonatomic materialization**, Replication Server selects rows at the primary table, without using a holdlock, and copies them over the network. Because the primary table is not locked, the replicate may go through visible steps that did not exist at the primary while nonatomic materialization is in progress.
- In nonmaterialization, the primary and replicate data is already in sync. You do not need to copy data over the network or load it from media. No updates can be in process while such a subscription is created.
- In **bulk materialization**, data is manually unloaded and loaded from media. This is the most efficient way to materialize subscriptions that involve a large amount of data.

For more information about subscription materialization methods, see the *Replication Server Administration Guide Volume 1*.

Atomic and nonatomic materialization commands

Use these commands to create a subscription and initialize data at the replicate database:

- create subscription – creates and materializes a subscription using atomic materialization.
- create subscription ... without holdlock – creates and materializes a subscription using nonatomic materialization.

If you use nonatomic materialization, which selects primary data without a holdlock, you must also use:

- set autocorrection – prevents failures caused by missing or duplicate rows in a replicate table. When primary data is selected without a holdlock, it might be updated before materialization is complete and before normal transaction replication begins.

Nonmaterialization command

Use this command to create a subscription when data is already in sync at the replicate database:

- create subscription ... without materialization – creates a subscription without materializing data at the replicate database.

Bulk materialization commands

Bulk materialization is used to manually coordinate subscription status and to transfer data for function replication definitions.

Use these commands for bulk materialization:

- define subscription – adds a subscription to the system tables at the primary and replicate Replication Server.
- activate subscription – starts the distribution of updates from the primary database to the replicate database and sets the subscription status to ACTIVE.

  After you use this command and verify status, manually load initial data from media into the replicate database. Use the with suspension option to prevent data from being applied to the replicate database until the load from media is complete.
- validate subscription – completes bulk materialization and changes the subscription status to VALID. Replication Server is notified that materialization is complete.

Other subscription commands

To monitor the materialization or dematerialization of a subscription, use:

- check subscription – finds the status of a subscription at the primary or replicate database.

To drop a subscription from a replicate database, use:

- drop subscription – clears subscription information from system tables.

Optionally, you can use drop subscription with purge to remove the replicate data associated with a subscription. This process is called **dematerialization**.

Publication subscription commands

Publication subscriptions use the same commands as subscriptions for replication definitions.

- To create a publication subscription using atomic materialization, nonatomic materialization, or nonmaterialization, use create subscription.
- To create a publication subscription using bulk materialization, use define subscription and the other bulk materialization commands.

When you add an article to a publication that has a subscription, you must refresh the publication subscription to include subscriptions for the new article. This process is called **rematerialization**.

- For atomic or nonatomic rematerialization, use create subscription with the for new articles clause.
- If data is in sync at the primary and replicate databases, use create subscription with the for new articles clause and the without materialization keywords.
- For bulk rematerialization, use define subscription with the for new articles clause, then use the other bulk materialization commands.

User commands

Users must have Replication Server login accounts to execute Replication Server commands. An account consists of a login name and a password, both of which must be supplied to connect to a Replication Server.

Use these commands to administer user login accounts:

- create user – adds a new user to a Replication Server.
- alter user – changes a user’s password.
- drop user – drops a Replication Server user account.

Use these commands to manage user permissions:

- grant – assigns permissions.
- revoke – revokes permissions.

Use the set proxy command to switch to another user login account with different permissions.

Each permission allows a user to execute a set of commands. For example, to create a replication definition, a user must have create object permission. A user with “sa” permission can execute any Replication Server command.

Database interface commands

Replication Server provides several ways to connect to databases and to customize the operations performed in them. Replication Server's open architecture supports primary or replicate databases managed by heterogeneous data servers, including Adaptive Server and several other data servers.

For each database, you can:

- Create or modify a Replication Server connection to a database. See "Database connection commands" on page 9 for details.
- Customize error handling methods. See "Error class commands" on page 10 for details.
- Customize database operations. See "Function and function string commands" on page 10 for details.
- Create or modify a logical database connection used in a warm standby application. See "Warm standby database commands" on page 12 for details.
- Set configuration parameters for the connection or logical connection. See "Configuration commands" on page 16 for details.

Each database that will be a source of replicated transactions or stored procedures must have a Replication Agent. For details, see the *Replication Server Administration Guide Volume 1*.

Database connection commands

A physical database **connection** connects a Replication Server to a local database that contains primary or replicate data. A Replication Server distributes messages to and from a database via a connection.

Use these commands to manage database connections:

- create connection – creates a database connection from Replication Server to a non-Sybase database. Adaptive Server database connections are added with rs_init.
- alter connection – changes or configures a database connection.
- drop connection – removes a database connection.
- suspend connection – suspends a database connection.

- resume connection – resumes a suspended connection.

Error class commands

An **error class** is a name under which error handling actions—such as retry and ignore—are assigned to specific data server errors.

Use the create connection command to associate an error class with a database. Use alter connection to change an error class. You can often create one error class for all databases for a specified data server.

Note The default error class for an Adaptive Server database is assigned when you add a connection using rs_init.

Use these commands to manage error handling actions and error classes:

- create error class – creates an error class.
- move primary – moves an error class or function-string class and any of the function-string class' derived classes to a different primary site.
- drop error class – drops an error class.
- assign action – assigns actions to data server error codes.

Use the stored procedure rs_init_erroractions to initialize a new error class created with error actions from an existing error class. For details, see Chapter 5, "Adaptive Server Commands and System Procedures."

Function and function string commands

You can use function strings to program Replication Server to execute customized commands at destination databases.

A **function** is a name associated with a data server operation. For example, rs_insert is the system function that inserts a row in a table, and rs_begin is the system function that initiates a transaction. System functions can manipulate data, as does rs_insert, or control transactions as does rs_begin.

Replication Server uses a template called a **function string** to construct the commands it submits to a database. At runtime, variables in the function string are replaced with values from the function.

A **function-string class** groups function strings for use with a database. For example, a function-string class might group all of the function strings for a vendor's data server or for a department's tables. Replication Server provides function-string classes for Adaptive Server and DB2 databases.

Use create connection to associate a function-string class with a database. Use alter connection to change a function-string class.

Note The default function-string class for Adaptive Server databases, rs_sqlserver_function_class, is assigned when you add a connection using rs_init.

You can create a new function-string class that inherits function strings from an existing class. Then you can customize only the function strings for which you want to specify non-default behavior, as your database or application requires.

Function-string class commands

Use these commands to work with function-string classes:

- create function string class – creates a function-string class.
- alter function string class – changes the inheritance relationships of a function-string class.
- move primary – moves an error class or function-string class and any of the function-string class' derived classes to a different primary site.
- drop function string class – drops a function-string class.

Function string commands

Use these commands to work with the function strings in a function-string class:

- create function string – creates a function string.
- alter function string – replaces an existing function string.
- drop function string – drops a function string.

Function commands

Use these commands to work with user-defined functions. They are only necessary for Asynchronous Procedure Calls.

- create function – creates a function string.
- drop function – drops a function.

Warm standby database commands

A Replication Server **warm standby application** maintains two Adaptive Server databases, one of which functions as a standby or backup copy of the other. Replication Server’s connection to the active and standby databases is called a **logical connection**.

Use these commands to manage logical database connections:

- create logical connection – creates a logical connection.
- alter logical connection – changes the characteristics of a logical connection.
- drop logical connection – removes a logical connection.
- configure logical connection – configures a logical connection.

Use these commands to perform tasks associated with warm standby applications:

- switch active – changes the active database.
- abort switch – aborts the switch active command, if possible.
- wait for switch – in an interactive or script-based Replication Server session, prevents commands from executing until the switch to a new active database is complete.
- wait for create standby – in an interactive or script-based Replication Server session, prevents Replication Server from accepting commands until the standby database is ready for operation.

Route commands

A **route** is a *one-way* message stream from the source (primary) Replication Server to the destination (target) Replication Server. A Replication Server sends messages to, or receives messages from, another Replication Server via a route. Such messages include data for replicated transactions. A route may connect Replication Servers across a local-area network or a wide-area network.

Use these commands to manage routes:

- create route – creates and configures a route from the current Replication Server to another.
- alter route – changes or reconfigures the route from the current Replication Server to another.
- drop route – removes the route to another Replication Server.
- suspend route – suspends the route to another Replication Server.
- resume route – resumes a suspended route.

System information commands

Use these commands to obtain information about the Replication Server. Any user can execute these commands.

- admin disk_space – displays the usage statistics of each disk partition accessed by the Replication Server.
- admin echo – returns the text you enter to verify that the Replication Server is running.
- admin get_generation – retrieves the generation number for a primary database.
- admin health – displays the overall status of the Replication Server.
- admin log_name – displays the path to the current log file.
- admin logical_status – displays status information for a logical connection in a warm standby application.
- admin pid – displays the process ID of the Replication Server.

- admin quiesce_check – determines if the queues in the Replication Server have been quiesced.
- admin quiesce_force_rsi – determines whether a Replication Server is quiesced and forces it to deliver outbound messages.
- admin rssd_name – displays the names of the data server and database for the Replication Server System Database (RSSD).
- admin security_property – displays network-based security mechanisms and features supported by Replication Server.
- admin security_setting – displays the status of network-based security features supported by Replication Server.
- admin set_log_name – closes the existing Replication Server log file and opens a new log file.
- admin show_connections – displays information about all connections from the Replication Server.
- admin show_function_classes – displays the names of existing function-string classes and their parent classes, and indicates the number of levels of inheritance.
- admin show_route_versions – displays the version number of routes that originate and terminate at the Replication Server.
- admin show_site_version – displays the site version of the Replication Server.
- admin sqm_readers – displays the read point and delete point for each Replication Server thread that is reading an inbound queue.
- admin stats – displays information and statistics about Replication Server counters.
- admin stats, backlog – reports the current transaction backlog in the stable queues.
- admin stats, {md | mem | mem_in_use} – reports information about memory usage.
- admin stats, status – displays the flushing status for all counters.
- admin stats, reset – resets all counters that can be reset.
- admin stats, {tps | cps | bps} – reports the number of transactions, commands, or bytes of throughput per second.
- admin time – displays the current time of Replication Server.

- admin translate – performs a datatype translation on a specific data value, displaying the results in literal format with delimiters.
- admin version – displays the Replication Server software version.
- admin who – displays information about threads running in the Replication Server.
- admin who_is_down – displays a subset of information about Replication Server threads that are not running.
- admin who_is_up – displays a subset of information about Replication Server threads that are running.

Partition commands

Replication Server stores messages in stable queues, which are stored on disk partitions. Inbound queues store messages received from Replication Agents; outbound queues store messages to be transmitted to data servers or other Replication Servers.

Use rs_init to create Replication Server's initial partitions. For more information about working with partitions in rs_init, see the Replication Server installation and configuration guides.

Use these commands to add or drop or change the size of partitions:

- create partition – makes a partition available to Replication Server. You must create a partition before you can add it.

  Note create partition replaces the existing add partition command. For backward compatibility, add partition is still supported as an alias for create partition but it will be depreciated in the future.

- drop partition – removes a partition from Replication Server.
- alter partition – changes the size of a partition.

For more information about stable queues and partitions, see the *Replication Server Administration Guide Volume 1*.

Configuration commands

When Replication Server starts, configuration parameters are read from system tables or from a configuration file. Configuration parameters may be static or dynamic. You can change dynamic parameters while Replication Server is running, but you must restart Replication Server after you change static parameters.

Use these commands to configure Replication Server:

- alter connection and configure connection – change the characteristics of a Replication Server connection to a database.
- configure logical connection – changes the Replication Server configuration for a logical connection in a warm standby application.
- configure replication server – changes Replication Server parameters and default parameters for routes and connections.
- alter route and configure route – change the characteristics of a route. A route connects one Replication Server to another.

Configuration parameters are also set when you create routes and connections using create route and create connection.

For more information, see the *Replication Server Administration Guide Volume 1*.

System administration commands

Use these commands to perform system administration tasks, and to troubleshoot problems that follow system failures. You must have “sa” permission to execute these commands.

Warning! Many of these commands should be used with caution and only in very restricted circumstances. Please check the associated documentation carefully before you use them.

- alter queue – specifies the behavior of a stable queue that encounters a large message of greater than 16K bytes. Use only if the Replication Server version is 12.5 or later and the site version is 12.1 or earlier.

- resume distributor – resumes a suspended distributor thread for a connection to a database.
- shutdown – shuts down a Replication Server.
- suspend distributor – suspends the distributor thread for a connection to a database.
- sysadmin apply_truncate_table – turns the “subscribe to truncate table” option on or off for existing subscriptions to a particular table, enabling or disabling replication of truncate table.
- sysadmin dropdb – drops references to a database from the ID Server.
- sysadmin dropldb – drops references to a logical database from the ID Server.
- sysadmin drop_queue – deletes a stable queue.
- sysadmin droprs – drops references to a Replication Server from the ID Server.
- sysadmin dump_file – specifies an alternate log file for use when a stable queue is dumped.
- sysadmin dump_queue – dumps the contents of a stable queue.
- sysadmin erssd – allows you to check ERSSD file locations and backup configurations, defragment ERSSD files, move ERSSD files or perform an unscheduled backup of the ERSSD.
- sysadmin fast_route_upgrade – updates the route version to Replication Server 15.0.1.
- sysadmin hibernate_off – turns off hibernation mode for the Replication Server and returns it to an active state.
- sysadmin hibernate_on – turns on hibernation mode for (or suspends) the Replication Server.
- sysadmin log_first_tran – writes the first transaction in a Data Server Interface (DSI) queue to the exceptions log.
- sysadmin purge_all_open – purges all open transactions from the inbound queue.
- sysadmin purge_first_open – purges the first open transaction from the inbound queue.
- sysadmin purge_route_at_replicate – removes all references to the primary Replication Server from a Replication Server at a replicate site.

- sysadmin restore_dsi_saved_segments – restores backlogged transactions so that they can be reapplied to the database.
- sysadmin set_dsi_generation – changes a database generation number in the RSSD to prevent Replication Server from reapplying transactions in the stable queue after a replicate database is restored.
- sysadmin site_version – sets the site version level.
- sysadmin sqm_purge_queue – removes all messages from a Replication Server Interface (RSI) stable queue.
- sysadmin sqm_unzap_command – restores a deleted message in a stable queue.
- sysadmin sqm_zap_command – deletes a single message in a stable queue.
- sysadmin sqt_dump_queue – dumps the transaction cache for each inbound or DSI queue.
- sysadmin system_version – sets the minimum Replication Server version level for the replication system.

Recovery commands

Use these commands to coordinate recovery after a database is reloaded or when Replication Server stable queues fail.

Warning! Many of these commands should be used with caution and only in very restricted circumstances. Make sure to check the associated documentation carefully before you use them.

- allow connections – places Replication Server in recovery mode for specified databases.
- ignore loss – allows Replication Server to accept messages after a loss is detected.
- rebuild queues – rebuilds Replication Server stable queues.
- resume log transfer – allows a RepAgent thread to connect to the Replication Server.

- resume queue – restarts a stable queue stopped after receiving a messager larger than 16K bytes. Applicable only when the Replication Server version is 12.5 or later and the site version is 12.1 or earlier.
- set log recovery – places Replication Server in log recovery mode for a database.
- suspend log transfer – disconnects a RepAgent from a Replication Server and prevents either from connecting.

For detailed recovery procedures, see the *Replication Server Administration Guide Volume 2*.

CHAPTER 2 Topics

This chapter contains information on these topics:

Datatypes

Replication Server directly supports the Sybase datatypes shown in Table 2-1.

Table 2-1: Replication Server-supported datatypes

Datatype class	Datatypes
Exact numeric (integer)	bigint, int, smallint, tinyint, unsigned bigint, unsigned int, unsigned smallint, unsigned tinyint, rs_address
Exact numeric (decimal)	decimal, numeric, identity
Approximate numeric (floating point)	float, real
Character	char(n), varchar(n), text
Money	money, smallmoney
Date/time	datetime, smalldatetime, date, time
Binary	binary(n), varbinary(n), image, rawobject, rawobject in row
Bit	bit
Unicode	unichar, univarchar, unitext
Java	rawobject, rawobject in row
Datatype definitions	See "Datatype definitions" on page 33.

RCL indirectly supports these Sybase datatypes:

- double precision
- nchar, nvarchar

These datatypes are not supported:

- The timestamp datatype

  A column defined with a timestamp datatype can only be updated by Adaptive Server.

  If you need to replicate data from a column that has a timestamp datatype in the primary table to a column in a replicate table, do not create the column in the replicate table with a timestamp datatype. If you do, when the replicate row is inserted, Adaptive Server generates error 272, "Can't update timestamp through rep server."

  To replicate a timestamp from the primary table, define a column on the replicate side as varbinary(8), and do the same in the replication definition.

  Another option is not to replicate the timestamp. Instead, add a column to the replicate table with a timestamp datatype, and allow Adaptive Server to fill that replicate table column when the replication occurs. Note that the column with the timestamp datatype on the replicate side will not match the timestamp on the primary side.

- The optional precision argument of the float datatype
- The optional precision and scale arguments of the exact decimal datatypes

Data in columns with unsupported datatypes can be replicated if you create the replication definition using one of the supported datatypes shown in Table 2-1. For example, to replicate a double precision column, define the column as float in the replication definition. To replicate a timestamp column, define the column as binary(8) in the replication definition. To replicate a column with a user-defined datatype, use the underlying datatype in the replication definition.

To replicate data stored in columns of type nchar or nvarchar in the Adaptive Server, use the char and varchar Replication Server datatypes, respectively. The only difference is that the length units in nchar and nvarchar refer to the number of characters in the native character set of the Adaptive Server, and the length units in char and varchar always refer to bytes.

To get the length of the corresponding Replication Server char and varchar datatypes, multiply the declared length of the nchar or nvarchar datatype by the value of the Adaptive Server global variable *@@ncharsize*.

For example, if *@@ncharsize* is 1 (true for all single-byte character sets like iso_1, cp850, cp437, roman8, and mac), there is a one-to-one correspondence and the declared lengths are the same. If *@@ncharsize* is 2 (true for some multibyte character sets like Shift-JIS and EUC-JIS), multiply the declared length of the nchar and nvarchar datatypes by 2 and declare them as char and varchar in the replication definition.

The following sections describe the supported datatypes. For more information about Adaptive Server datatypes, see the *Adaptive Server Enterprise Reference Manual*.

Exact numeric (integer) datatypes

Replication Server supports these exact numeric (integer) datatypes:

- bigint – whole numbers between -2^{63} and $+2^{63} - 1$ (-9,233,372,036,854,775,808 and +9,233,372,036,854,775,807), inclusive
- int – whole numbers between -2^{31} and $+2^{31} - 1$ (-2,147,483,648 and 2,147,483,647), inclusive
- smallint – whole numbers between -2^{15} and $+2^{15} - 1$ (-32,768 and -32,767), inclusive
- tinyint – positive whole numbers between 0 and 255, inclusive
- unsigned bigint – whole numbers between 0 and 18,446,744, 073, 709,551,615, inclusive
- unsigned int – whole numbers between 0 and 4,294,967,295, inclusive
- unsigned smallint – whole numbers between 0 and 65535, inclusive
- unsigned tinyint – whole numbers between 0 and 255, inclusive.

The rs_address datatype, which uses the underlying datatype int, is used in a special method of subscription resolution. For more information about the rs_address datatype, see create subscription. See also the *Replication Server Administration Guide Volume 1.*

Exact numeric (decimal) datatypes

Replication Server supports the following exact numeric (decimal) datatypes:

decimal – exact decimal numbers between -10^{38} and 10^{38} -1, inclusive.

numeric – exact decimal numbers between -10^{38} and 10^{38} -1, inclusive.

When you create a replication definition, omit the length and precision from numeric datatype declarations. Replication Server processes numeric values without affecting precision.

Identity columns use numeric underlying datatype, with exact decimal numbers of scale 0 between 1 and 10^{38} -1, inclusive.

When you create a replication definition for a table that contains an identity column, specify "identity" as the datatype for the column.

This command is applied to the replicated table *before* an insert command:

```
set identity_insert table_name on
```

This command is applied to the replicated table *after* an insert command:

```
set identity_insert table_name off
```

Identity columns are never updated by the update command.

If the replicate data server is Adaptive Server and a table contains an identity column, the maintenance user must be the owner of the table (or must be the "dbo" user or aliased to the "dbo" login name) at the replicate database in order to use the Transact-SQL identity_insert option.

Approximate numeric (floating point) datatypes

There are two approximate numeric (floating point) datatypes:

- float – positive or negative floating point numbers. Precision and number of significant digits are machine-dependent. Storage size is 8 bytes.
- real – like float except the storage size is 4 bytes.

Character datatypes

Note The Unicode datatypes unichar, univarchar, and unitext have the same attributes as their char, varchar, and text equivalents. See "Unicode datatypes" on page 30.

- char(n) – any combination of up to 32,768 single-byte letters, symbols, and numbers. Specify the maximum size of the string with *n*. A char value can contain 0 characters, but *n* must be between 1 and 32,768. A multibyte string cannot exceed 32,768 bytes.
- varchar(n) – any combination of up to 32,768 single-byte letters, symbols, and numbers. Specify the maximum size of the string with *n*. A varchar value can contain 0 characters if it is defined to allow null values, but *n* must be between 1 and 32,768. A multibyte string cannot exceed 32,768 bytes.

The difference between char and varchar data is the way the values are stored in Adaptive Server databases. Replication Server treats them as equivalent types, but maintains the distinction so that the storage method is the same in primary and replicate databases.

- text – variable-length character columns up to 2,147,483,647 bytes in length. Replication Server does not support datatype conversion of text data. This means if your replication definition declares a column as text, the primary table's columns must be text.

Entry format for character data

Literal char, varchar, and text values—or their equivalents—must be enclosed in single quotation marks.

You can embed single quotation marks in char and varchar literals in two ways. Use two consecutive quotation marks to represent a single embedded quotation mark, as in this example:

```
'''You can have cake if you bake it,'' Ed claims.'
```

The first and last quotation marks delimit the character string. The two internal pairs of quotation marks are interpreted as embedded single quotation marks.

Replication Server generates single quotation marks when it substitutes a character value for a variable in a function-string template. For more information, see create function string.

Money datatypes

The money datatypes hold fixed precision values for currency or monetary values:

- money – monetary values between --922,337,203,685,477.5808 and 922,337,203,685,477.5807, with accuracy to 1/10000 of a monetary unit. Storage size is 8 bytes.
- smallmoney – monetary values between -214,748.3648 and 214,748.3647, with accuracy to 1/10000 of a monetary unit. Storage size is 4 bytes.

Entry format for money data

Precede money and smallmoney literal values with a U.S. dollar sign ($) to distinguish them from the floating point datatypes. For negative values, place the minus sign after the dollar sign.

Replication Server outputs a dollar sign when it substitutes money and smallmoney values into function-string output templates.

Date/time, and date and time datatypes

Replication Server supports four datatypes for date and time data:

- datetime – dates and times of day between January 1, 1753 and December 31, 9999. Storage size is 8 bytes: 4 bytes for the number of days before or after the base date of January 1, 1900, and 4 bytes for the time, to 1/300 second. Dates before the base date are stored as negative values.
- smalldatetime – dates and times of day between January 1, 1900 and June 6, 2079, with accuracy to one minute. Storage size is 4 bytes: one small integer for the number of days after January 1, 1900, and one small integer for the number of minutes since midnight.
- date – dates between January 1, 0001, and December 31, 9999. Storage size is 4 bytes. Dates before the base date are stored as negative values.

Entry format for date/time values

Enter datetime and smalldatetime values as character strings, enclosed in single quotation marks.

Replication Server encloses datetime values in single quotation marks when it substitutes datetime values into function-string output templates. Be sure to consider this when you create function strings that include datetime variables.

The date and time portions of the data are recognized separately; therefore, the time can precede or follow the date. If you omit the time, Replication Server assumes midnight (12:00:00:000AM). If you omit the date, Replication Server assumes January 1, 1900.

Enter times according to these general rules:

- Hours range from 0 to 23; minutes and seconds range from 0 to 59; milliseconds range from 0 to 999.
- A value must have a colon or an “AM” or “PM” indicator to be recognized as a time value.
- You can append “AM” or “PM,” with or without an intervening space. 12AM is midnight and 12PM is noon. If you specify AM, the hour must be between 1 and 12 (0 is acceptable in place of 12). If you specify PM, the hour must be between 13 and 23.
- Milliseconds can be preceded by either a colon or a period. If preceded by a colon, the number means thousandths of a second. If preceded by a period, a single digit means tenths of a second, two digits mean hundredths of a second, and three digits mean thousandths of a second. For example, “12:30:20:1” means twenty and one-thousandth of a second past 12:30; “12:30:20.1” means twenty and one-tenth of a second past 12:30.
- You can omit any portion of a time value. If you omit seconds, you must also omit milliseconds. If you omit minutes, you must also omit seconds and milliseconds. Replication Server assumes zero for any omitted part.

Here are some examples of time literals:

```
2:00
14.30
14:30:20
14:30:20:500
4pm
11:41:36 AM
12:48:5.333 pm
```

Enter dates with the month, day, and year in any order, subject to the following rules:

- You can enter the month as a number from 1–12, or use the U.S. English month name or its three-character abbreviation.
- If you use the numeric month, the date parts must be separated with slashes (/), hyphens (-), or periods (.). The date parts must be given in month-day-year order.

- These examples show different ways to enter the date March 15, 1998:

  ```
  3-15-1998
  March-15-1998
  March 15 1998
  15/March/1998
  March.15.1998
  ```

- You can abbreviate U.S. English months to 3 characters. Case is not significant.

  ```
  JAN 9 1998
  31 oct 1997
  ```

- When you use an alphabetic month, the month and day can be followed by a comma. These are valid dates:

  ```
  Nov 17, 1997
  1997 Nov, 17,
  17 Nov, 1997
  ```

- You can enter the year with one, two, or four digits. A one- or two-digit year less than 50 is assumed to be in the next (twenty-first) century. A two-digit year greater than or equal to 50 is in the current (twentieth) century.
- Four-digit years are recognized anywhere in a date value. Two-digit years must appear after the day of the month.
- You can omit the day of the month if you use the alphabetic month and a four-digit year. The day defaults to the first of the month. You cannot use separators other than commas after the month name.

  Replication Server interprets these dates as May 1, 1998:

  ```
  May 1998
  1998 MAY
  may, 1998
  ```

Binary datatypes

The binary datatypes are:

- binary(n) – up to 32,768 bytes of fixed-length binary data. The binary datatypes are used for storing programming code or pictures, not for numeric values. Specify the maximum byte length of the value with *n*. A binary value can contain 0 bytes, but *n* must be between 1 and 32,768.

- varbinary(n) – up to 32,768 bytes of variable-length binary data. The varbinary datatypes are used for storing programming code or pictures, not for numeric values. Specify the maximum byte length of the value with *n*. A varbinary value can contain 0 bytes, but *n* must be between 1 and 32,768.

  The difference between binary and varbinary data is the way the values are stored in Adaptive Server databases. Replication Server treats them as equivalent types, but maintains the distinction so that the storage method is the same in primary and replicate databases.

- rawobject in row – 255 bytes of variable-length binary data. The rawobject in row datatype is used to store serialized Java values within the data pages allocated to the table.

  Replication Server handles rawobject in row data exactly as it handles varbinary data. The base datatype for rawobject in row is varbinary(255). See also "Java datatypes" on page 32.

- rawobject large in row – 32,768 bytes of variable-length binary data. The rawobject large in row datatype is used to store serialized Java values within the data pages allocated to the table.

  Replication Server handles rawobject large in row data the same as it handles varbinary data. The base datatype for rawobject large in row is varbinary(32768). See also "Java datatypes" on page 32.

- image – variable-length binary columns up to 2,147,483,647 bytes in length. Replication Server does not support datatype conversion of image data.

- rawobject – variable-length binary columns up to 2,147,483,647 bytes in length. The rawobject datatype is used to store serialized Java values. Replication Server does not support datatype conversion of rawobject data. This means if your replication definition declares a column as rawobject, the primary table's column must be rawobject.

  Replication Server handles rawobject data exactly as it handles image data. The base datatype for rawobject is image. See also "Java datatypes" on page 32.

Entry format for binary data

Enter binary, varbinary, image, rawobject, and rawobject in row literal values using the hexadecimal digits 0-9 and A-F (or a-f). Each byte is represented by 2 hexadecimal digits, and the entire value is preceded by "0x". The following example is a 10-byte binary string:

```
0x010305070B0D1113171D
```

Replication Server outputs the “0x” prefix when it substitutes binary values in function-string output templates.

Bit datatype

The bit datatype is used for Boolean values:

bit – either 1 or 0. Integer values other than 1 or 0 are interpreted as 1.

Unicode datatypes

Replication Server supports three Unicode datatypes, unichar, univarchar, and unitext. Unicode allows you to mix languages from different language groups in the same data server.

The Unicode datatypes behave exactly like their equivalent Replication Server datatypes. See “Character datatypes” on page 24 for more information.

- unichar –> char
- univarchar –> varchar
- unitext –> text

The Unicode datatypes share the syntax and semantics of their equivalent datatypes, except Unicode values are always stored in UTF-16, regardless of the Replication Server default character set. unichar is a fixed-width, non-nullable datatype. univarchar is a variable-width, nullable datatype. unitext is variable-width, nullable datatype.

You can:

- Replicate unichar, univarchar, and unitext columns to replicate and standby databases
- Use unichar and univarchar columns in the primary key of a replication definition
- Use unichar and univarchar columns as searchable columns in a replication definition and in the where clauses of associated subscriptions and articles
- Use unichar and univarchar columns as searchable columns in a function replication definition and in the where clauses of associated subscriptions and articles

- Use unichar, univarchar, and unitext columns when replicating to or from heterogeneous data servers

In the same way as text:

- unitext columns cannot be part of the primary key in the replication definition.
- unitext columns cannot be specified as searchable columns in a replication definition.
- unitext columns cannot be specified as searchable columns in a function replication definition.
- unitext datatype cannot be used as a base datatype or a datatype definition or as a source or target of either a column-level or class-level translation.

To correctly replicate the unichar and univarchar columns, the Replication Server must be configured:

```
RS_charset=utf8
```

If the Replication Server default character set is not UTF-8, Replication Server can replicate only unichar and univarchar characters in the ASCII-7 code range.

Upgrade issues

To fully support the unichar and univarchar datatypes, both the primary and replicate Replication Server must be running version 12.5 or later.

To fully support the unitext datatype, both the primary and replicate Replication Servers must be running version 15.0.1 or later, the route version must be 15.0.1, and the LTL version must be 700. If the LTL version is less than 700 at connect-source time, RepAgent converts unitext columns to image.

The RM route upgrade feature copies replication definitions referencing unichar, univarchar, and unitext datatypes from upstream Replication Servers.

Mixed-version issues

In a mixed-version environment, the route version between the primary and replicate Replication Servers determines which features are supported.

- A replication definition created with unichar or univarchar columns is not propagated to Replication Server version 12.1 and earlier.

  A replication definition created with unitext columns is not propagated to Replication Server version 12.6 and earlier.

- A replication definition subscribed by Replication Server version 12.1 and earlier cannot be altered to add unichar or univarchar.

  A replication definition subscribed by Replication Server version 12.6 and earlier cannot be altered to add unitext columns.

- If a replication definition is altered to add unichar or univarchar columns, it is dropped from 12.1 and earlier versions of downstream Replication Servers, and the oldest replication definition that is compatible with version 12.1 and earlier is propagated.

- A replication definition created with unichar or univarchar columns is propagated to Replication Server version 12.1 or earlier if the unichar or univarchar columns are removed.

  Similarly, a replication definition created with unitext columns is propagated to Replication Server version 12.6 or earlier if the unitext columns are removed.

Java datatypes

Java columns pass through the replication system as any of three Replication Server datatypes:

- As rawobject, in which the information is stored in the database in a separate location in the same way that image data is stored. The base datatype of rawobject is image. rawobject is the default datatype for Java columns in Replication Server.

- As rawobject in row, in which the information is stored in the database on consecutive data pages allocated to the table in the same way that char data is stored. The base datatype of rawobject in row is varbinary(255).

- As rawobject large in row, in which the information is stored in the database on consecutive data pages allocated to the table in the same way that char data is stored. The base datatype of rawobject large in row is varbinary(32768).

rawobject, rawobject in row, and rawobject large in row datatypes are compatible only with their base datatypes. They are not compatible with each other. You cannot replicate one Java datatype to the other Java datatype, or vice versa. See also “Binary datatypes” on page 28.

The rs_subcmp reconciliation utility treats Java datatypes as their base datatypes.

Datatype definitions

Sybase provides a set of user defined datatypes and datatype classes. You can use them to change the datatype of column values when you replicate between:

- Sybase data servers
- Sybase data servers and non-Sybase data servers
- Homogeneous non-Sybase data servers
- Heterogeneous non-Sybase data servers

A datatype definition describes a non-Sybase datatype in terms of a base Replication Server native datatype. The base datatype determines the maximum and minimum length associated with the datatype definition and provides defaults for other datatype attributes. The base datatype also defines the delimiters associated with the datatype definition.

Each datatype class contains datatype definitions for a specific data server. The datatype classes are:

- Adaptive Server – rs_sqlserver_udd_class
- Adaptive Server Anywhere – rs_asa_udd_class
- DB2 – rs_db2_udd_class
- Informix – rs_informix_udd_class
- Microsoft SQL Server – rs_msss_udd_class
- Oracle – rs_oracle_udd_class

For a list and description of supported datatype definitions for each datatype class, see the *Replication Server Heterogeneous Replication Guide*.

Identifiers

Identifiers are symbolic names for objects—databases, tables, replication definitions, publications, subscriptions, functions, parameters, function string variables, and so on.

Identifiers are 1–255 bytes long for these objects:

- Tables
- Columns

- Procedures
- Parameters
- Functions - as part of function replication definition or internal functions

---

Note The create function, alter function, and drop function commands do not support long identifiers. The name of the function and the parameters of these commands cannot exceed 30 bytes.

- Function strings
- Replication definitions – including table replication definitions, function replication definitions, and database replication definitions
- Articles
- Publications

All other identifiers are 1–30 bytes long.

If an identifier is not enclosed in quotes, its first character must be an ASCII letter. Subsequent characters can be ASCII letters, digits, or the $ or _ character. Embedded spaces are not allowed.

Identifiers that begin with the characters “rs_” are reserved for Replication Server. See “Reserved words” on page 36 for a list of other reserved words.

Parameter names for Replication Server functions and Adaptive Server stored procedures are the only identifiers that can begin with the @ character.

- Replication Server function parameter names can be up to 256 bytes including the @ character.
- Adaptive Server stored procedure parameter names can be up to 255 bytes including the @ character.

You can use reserved words for identifiers by enclosing the identifiers in double quotes. When you use quotes, you can also use embedded spaces and otherwise prohibited characters, such as !@#$%^&*(), and 8-bit and multibyte characters. Replication Server strips any trailing blanks from the end of the identifier, even if you have placed it within quotes. For example:

```
check subscription "publishers_sub"
     for "publishers_rep"
```

```
with replicate at "SYDNEY_DS"."pubs2"
```

Warning! Adaptive Server allows you to place identifiers within quotes when you set quoted_identifier to on. This lets you use reserved words for Adaptive Server object names. However, Replication Server does not recognize identifiers in quotes in the commands that it sends to Adaptive Server, so you cannot use Transact-SQL keywords as names for replicated Adaptive Server objects.

If necessary, you can alter function strings to place quotes around identifiers for replicated objects.

Enclose variable names in function-string templates in question marks. For example, this variable name could be used in a function string to refer to a primary database:

```
?rs_origin_db!sys?
```

or, using quoted identifiers:

```
?”rs_origin_db”!sys?
```

Name space for identifiers

The name space of an identifier is the scope in which Replication Server recognizes it. A data server name, for example, has a global name space because the name can be used for only one data server in the entire replicated data system. A column name, on the other hand, has table scope; it must be qualified with the name of the table because more than one table can have a column with the same name.

Table 2-2 shows the Replication Server name space for each identifier.

Table 2-2: Name space for Replication Server identifiers

Identifier type	Name space
Article	Publication
Column	Table
Data server	Global
Database	Data server
Error class	Global
Function-string class	Global

Identifier type	**Name space**
Function	Replication definition. User-defined functions used for asynchronous procedures executed in Adaptive Server databases must have globally unique names, unless a table replication definition is specified in the procedure.
Function replication definition	Global
Parameter	Function
Publication	Primary data server and database
Replication definition	Global
Replication Server	Global
Subscription	Replication definition, replicate data server, and database. Subscriptions must have globally unique names.
User	Replication Server
Variable	Function or table

You should adopt a naming convention for replication definitions and other Replication Server objects with global scope to ensure that names remain unique in the global name space.

Warning! Identifiers with global name space must be managed carefully. Replication Server cannot detect all duplications in the global name space immediately, but errors may occur later.

Identifiers with a name space other than global sometimes must be qualified. For example, the syntax for many Replication Server commands includes an at clause, which identifies the data server and database where a table is located:

at *data_server.database*

In a correctly configured system, all servers will use the same sort order. If servers do not use the same sort order, different servers will compare identifiers inconsistently, which can lead to abnormal behavior in the network.

Reserved words

The words in Table 2-3 are reserved Replication Server keywords. Although the words are shown in lowercase, Replication Server is not case-sensitive. Therefore, all combinations of uppercase and lowercase letters are reserved. Replication Server also reserves all keywords and identifiers beginning with “rs_”.

Table 2-3: Replication Server reserved words

abort	action	activate	active
add	admin	_af	after
all	allow	alter	always_rep
always_replicate	and	_ap	article
articles	_apd	append	applied
_ar	as	assign	at
before	begin	_bf	_bg
changed	_ch	check	ci
class	_cm	columns	commit
configure	connect	connection	connections
controller	create	database	datarow
ddl	debug	define	definition
deliver	description	disconnect	distribute
distribution	distributor	_dr	drop
drop_repdef	_ds	dsi_suspended	dump
enable	error	exec	execute
expand	_fi	first	for
from	function and functions	get	grant
_ha	hastext	holdlock	ignore
in	incrementally	internal_use_only	into
installjava	_instj	_isb	_isbinary
_jar	key	language	large
last	load	log	logical
loss	maintenance	map	marker
materialization	message	_mbf	min_before
minimal	min_row	move	_mr
name	named	_ne	never_rep
new	next	no	no_password
none	not	notrep	nowait
npw	_nr	_nu	null
nullable	of	off	on
only	open_xact	or	osid
output	overwrite	owner	parameters
parent	partition	passthru	password
primary	procedure	procedures	public
publication	purge	queue	queues
_rc	rebuild	reconfigure	recover

recovery	reject	remove	repfunc
replay	rep_if_changed	replicate	replicate_if_changed
replication	resume	retry	revoke
_rf	_rl	rollback	route
row	rpc	rs_ticket	rsrpc
scan	searchable	segment	select
send	sendallxacts	seq	server
set	shutdown	site	size
skip	source	sqlddl	standby
starting	string	subscribe	subscription
suspend	suspension	switch	sysadmin
sys_sp	system	table and tables	textcol
textlen	_tl	_tn	to
_tp	tpinit	tpnull	_tr
trace	tran	transaction and transactions	transfer
truncate	truncation	twosave	_up
unsigned	update	use	user
username	using	validate	verify
vers	wait	warmstdb	_wh
where	with	without	writetext
_yd	yielding	zerolen	_zl

Support for Adaptive Server

This section outlines specific Replication Server support for Adaptive Server.

Replication Server supports international customers by providing:

- Support for all Sybase-supported character sets, including 8-bit, multibyte character sets, and Unicode character sets
- Support for all Sybase-supported sort orders, including non-binary sort orders and Unicode sort orders
- Localization of Replication Server messages into English, French, German, and Japanese languages

- Support for Replication Server logical page size, number of columns and columns size, number of arguments for stored procedures

The following information describes these features. For guidelines on designing a replication system in an international environment, refer to Chapter 7, "International Replication Design Considerations," in the *Replication Server Design Guide*.

Character set support

Replication Server supports all Sybase-supported character sets and performs character set conversion of data and identifiers, as needed. The following guidelines apply to character set conversion:

- Replication Server, like all Sybase software, cannot convert between single-byte character data and multibyte character data.
- Identifiers, such as table and column names, that contain multibyte characters or single-byte characters with the high bit set must be enclosed in double quotes.

Specifying character sets

You specify character sets with the *RS_charset* parameter in the Replication Server configuration file. You can also specify a character set for writing the Replication Server configuration file. This parameter is CONFIG_charset.

For replication to work properly, the Replication Server's character set must be the same as the character set of the data servers it controls. It should also be compatible with the character sets of all other Replication Servers in your system.

Character set conversion

Replication Server performs character set conversion on data and identifiers between primary and replicate databases. However, Replication Server does not perform character set conversion between incompatible character sets. If the character sets are compatible, but one or more characters are not common to both character sets, a question mark (?) is substituted for the unrecognized characters.

A configuration parameter in the rs_config system table, dsi_charset_convert, gives you options for how Replication Server handles character set conversion. You set this parameter with the alter connection command. For more information about these options, see alter connection on page 108.

rs_get_charset system function

Each time Replication Server connects to a data server, it executes rs_get_charset, which obtains the character set used by the data server. If it is not what is expected, Replication Server prints a warning message to the error log file. For more information, see rs_get_charset on page 372.

Sort order support

Replication Server uses sort order, or collating sequence, to determine how character data and identifiers are compared and ordered. Replication Server supports all Sybase-supported sort orders, including non-binary sort orders. Non-binary sort orders are necessary for the correct ordering of character data and identifiers in European languages.

You specify sort orders with the *RS_sortorder* parameter in the Replication Server configuration file. You can specify any Sybase-supported sort order that is compatible with your character set.

For replication to work properly, all sort orders in your replication system should be the same.

rs_get_sortorder system function

Each time Replication Server connects to a data server, it executes rs_get_sortorder, which obtains the sort order used by the data server. If it is not what is expected, Replication Server prints a warning message to the error log file. For more information, see rs_get_sortorder on page 375.

Message language support

Replication Server can print messages in French, German, and Japanese to the error log and to clients. You specify languages with the *RS_language* parameter in the Replication Server configuration file.

You can specify any language to which the Replication Server has been localized that is compatible with your character set. English is the default language and is compatible with all Sybase character sets.

Stored procedure messages

The rs_msgs system table stores localized error messages used during installation and by the Replication Server stored procedures that manage the RSSD. For details about this system table, see rs_queuemsg.

Extended page- and column-size support

Replication Server version 12.5 and later supports the extended limits supported by Adaptive Server version 12.5 and later. Replication Server supports:

- A choice of logical page sizes: 2K, 4K, 8K, or 16K
- Larger rows (to the limit of the page size)
- Wider columns (to the limit of the page size)
- Wider index keys
- More columns per table
- Larger messages (greater than 16K bytes)

For more information about extended limits in Replication Server, see the *Replication Server Administration Guide Volume 1*.

CHAPTER 3 Replication Server Commands

This chapter contains the reference pages for the RCL commands.

Table 3-1 provides a brief description of the commands in this chapter.

Table 3-1: RCL commands

Command	Description
abort switch	Aborts the switch active command, unless Replication Server has gone too far in the active switch process to abort it.
activate subscription	For a subscription to a replication definition or a publication, starts the distribution of updates from the primary to the replicate database and sets the subscription status to ACTIVE.
add partition	Makes a partition available to Replication Server. A partition can be a disk partition or an operating system file. See create partition.
admin config	Retrieves Replication Server parameters such as global, connection, logical connection, and route parameters.
admin disk_space	Displays use of each disk partition accessed by the Replication Server.
admin echo	Returns the string entered by the user.
admin get_generation	Retrieves the generation number for a primary database.
admin health	Displays the status of the Replication Server.
admin log_name	Displays the path to the current log file.
admin logical_status	Displays status information for logical connections.
admin pid	Displays the process ID of the Replication Server.
admin quiesce_check	Determines if the queues in the Replication Server have been quiesced.
admin quiesce_force_rsi	Determines whether a Replication Server is quiescent and forces it to deliver and obtain acknowledgments for messages in RSI queues.
admin rssd_name	Displays the names of the data server and database for the RSSD.
admin security_property	Displays information about supported network-based security mechanisms and security services.
admin security_setting	Displays network-based security parameters and values for the Replication Server.
admin set_log_name	Closes the existing Replication Server log file and opens a new log file.
admin show_connections	Displays information about all connections from the Replication Server to data servers and to other Replication Servers.
admin show_function_classes	Displays the names of existing function-string classes and their parent classes, and indicates the number of levels of inheritance.

Command	Description
admin show_route_versions	Displays the version number of routes that originate at the Replication Server and routes that terminate at the Replication Server.
admin show_site_version	Displays the site version of the Replication Server.
admin sqm_readers	Displays the read and delete points of the threads that are reading a stable queue.
admin stats	Displays information and statistics about Replication Server counters.
admin stats, backlog	Reports the current transaction backlog in the stable queues.
admin stats, canceladmin stats, cancel	Cancels the currently running asynchronous command.
admin stats, { md | mem | mem_in_use }	Reports information about memory usage.
admin stats, reset	Resets all counters that can be reset.
admin stats, status	Displays the flushing status for all counters.
admin stats, { tps | cps | bps }	Reports the number of transactions, commands, or bytes of throughput per second.
admin time	Displays the current time of Replication Server.
admin translate	Performs a datatype translation on a value, displaying the results in delimited literal format.
admin version	Displays the version number of the Replication Server software.
admin who	Displays information about threads running in the Replication Server.
admin who_is_down	Displays information about Replication Server threads that are not running.
admin who_is_up	Displays information about Replication Server threads that are running.
allow connections	Places Replication Server in recovery mode for specified databases.
alter connection	Changes the attributes of a database connection.
alter database replication definition	Changes an existing database replication definition.
alter function	Adds parameters to a user-defined function.
alter function replication definition	Changes an existing function replication definition.
alter function string	Replaces an existing function string.
alter function string class	Alters a function-string class, specifying whether it should be a base class or a derived class.
alter logical connection	Disables or enables the Distributor thread for a logical connection, changes attributes of a logical connection, and enables or disables replication of truncate table to the standby database.
alter partition	Alters the size of a partition.
alter queue	Specifies the behavior of the stable queue that encounters a large message of greater than 16K bytes. Applicable only when the Replication Server version is 12.5 or later and the site version is 12.1 or earlier.
alter replication definition	Changes an existing replication definition.

Command	Description
alter route	Changes the attributes of a route from the current Replication Server to a remote Replication Server.
alter user	Changes a user’s password.
assign action	Assigns Replication Server error-handling actions to data server errors received by the DSI thread.
check publication	Finds the status of a publication and the number of articles the publication contains.
check subscription	Finds the materialization status of a subscription to a replication definition or a publication.
configure connection	Changes the attributes of a database connection.
configure logical connection	Changes attributes of a logical connection.
configure replication server	Sets characteristics of the Replication Server, including network-based security.
configure route	Changes the attributes of a route from the current Replication Server to a remote Replication Server.
create article	Creates an article for a table or function replication definition and specifies the publication that is to contain the article.
create connection	Adds a database to the replication system and sets configuration parameters for the connection. To create a connection for an Adaptive Server database, use Sybase Central or rs_init.
create database replication definition	Creates a replication definition for replicating a database or a database object.
create error class	Creates an error class.
create function	Creates a user-defined function.
create function replication definition	Creates a function replication definition and user-defined function for a stored procedure that is to be replicated.
create function string	Adds a function string to a function-string class. Replication Server uses function strings to generate instructions for data servers.
create function string class	Creates a function-string class.
create logical connection	Creates a logical connection. Replication Server uses logical connections to manage warm standby applications.
create partition	Makes a partition available to Replication Server. A partition can be a disk partition or an operating system file.
create publication	Creates a publication for tables or stored procedures that are to be replicated as a group to one or more subscribing replicate databases.
create replication definition	Creates a replication definition for a table that is to be replicated.
create route	Designates the route to use for a connection from the current Replication Server to a remote Replication Server.
create subscription	Creates and initializes a subscription and materializes subscription data. The subscription may be for a database replication definition, a table replication definition, a function replication definition, or a publication.

Command	Description
create user	Adds a new user login name to a Replication Server.
define subscription	Adds a subscription to the Replication Server system tables, but does not materialize or activate the subscription. The subscription may be for a database replication definition, a table replication definition, a function replication definition, or for a publication. This command begins the process of bulk subscription materialization, or the process of refreshing a publication subscription.
drop article	Drops an article and optionally drops its replication definition.
drop connection	Removes a database from the replication system.
drop database replication definition	Drops an existing database replication definition.
drop error class	Drops an error class and any actions associated with it.
drop function	Drops a user-defined function and its function strings.
drop function replication definition	Drops a function replication definition and its user-defined function.
drop function string	Drops a function string for a function-string class.
drop function string class	Drops a function-string class.
drop logical connection	Drops a logical connection. Logical connections are used to manage warm standby applications.
drop partition	Removes a disk partition from the Replication Server.
drop publication	Drops a publication and all of its articles, and optionally drops the replication definitions for the articles.
drop replication definition	Drops a replication definition and its functions.
drop route	Closes the route to another Replication Server.
drop subscription	Drops a subscription to a database replication definition, table replication definition, function replication definition, article, or publication.
drop user	Drops a Replication Server user login name.
grant	Assigns permissions to users.
ignore loss	Allows Replication Server to accept messages after it detects a loss.
move primary	Changes the primary Replication Server for an error class or a function-string class.
rebuild queues	Rebuilds Replication Server stable queues.
resume connection	Resumes a suspended connection.
resume distributor	Resumes a suspended Distributor thread for a connection to a database.s
resume log transfer	Allows the RepAgent to connect to the Replication Server.
resume queue	Restarts a stable queue stopped after being passed a message larger than 16K bytes. Applicable only when the Replication Server version is 12.5 or later and the site version has not been similarly upgraded.
resume route	Resumes a suspended route.

Command	Description
revoke	Revokes permissions from users.
set autocorrection	Prevents failures that would otherwise be caused by missing or duplicate rows in a replicated table.
set log recovery	Specifies databases whose logs are to be recovered from offline dumps.
set proxy	Switches to another user.
shutdown	Shuts down a Replication Server.
suspend connection	Suspends a connection to a database.
suspend distributor	Suspends the Distributor thread for a connection to a primary database.
suspend log transfer	Disconnects a RepAgent from a Replication Server and prevents a RepAgent from connecting.
suspend route	Suspends a route to another Replication Server.
switch active	Changes the active database in a warm standby application.
sysadmin apply_truncate_table	Turns on or off the “subscribe to truncate table” option for all existing subscriptions to a particular table, enabling or disabling replication of truncate table.
sysadmin dropdb	Drops a database from the ID Server.
sysadmin dropldb	Drops a logical database from the ID Server.
sysadmin drop_queue	Deletes a stable queue. Use this command to drop a failed materialization queue.
sysadmin droprs	Drops a Replication Server from the ID Server.
sysadmin dump_file	Specifies an alternative log file name for use when dumping a Replication Server stable queue.
sysadmin dump_queue	Dumps the contents of a Replication Server stable queue.
sysadmin erssd	Displays ERSSD name, schedule, backup directory, and ERSSD file locations. Used with options, this command performs unscheduled backups and moves ERSSD files.
sysadmin fast_route_upgrade	Updates the route version to the site version of the lower of the primary or replicate Replication Server.
sysadmin hibernate_off	Turns off hibernation mode for the Replication Server and returns it to an active state.
sysadmin hibernate_on	Turns on hibernation mode for (or suspends) the Replication Server.
sysadmin log_first_tran	Writes the first transaction in a DSI queue into the exceptions log.
sysadmin purge_all_open	Purges all open transactions from an inbound queue of a Replication Server.
sysadmin purge_first_open	Purges the first open transaction from the inbound queue of a Replication Server.
sysadmin purge_route_at_replicate	Removes all references to a primary Replication Server from a replicate Replication Server.
sysadmin restore_dsi_saved_segments	Restores backlogged transactions.

Command	Description
sysadmin set_dsi_generation	Changes a database generation number in the Replication Server to prevent the application of transactions in the DSI stable queue after a replicate database is restored.
sysadmin site_version	Sets the site version number for the Replication Server. This lets you use the software features in the corresponding release, and prevents you from downgrading to an earlier release.
sysadmin sqm_purge_queue	Purges all messages from a stable queue.
sysadmin sqm_unzap_command	Undeletes a message in a stable queue.
sysadmin sqm_zap_command	Deletes a single message in a stable queue.
sysadmin sqt_dump_queue	Dumps the transaction cache for an inbound queue or a DSI queue.
sysadmin system_version	Displays or sets the system-wide version number for the replication system, allowing you to use the software features in the corresponding release level.
validate publication	Sets the status of a publication to VALID, allowing new subscriptions to be created for the publication.
validate subscription	For a subscription to a replication definition or a publication, sets the subscription status to VALID. This command is part of the bulk materialization process, or part of the process of refreshing a publication subscription.
wait for create standby	A blocking command that allows a client session in the Replication Server to wait for the standby database creation process to complete.
wait for delay	Specifies a time interval at which this command is blocked.
wait for switch	A blocking command that allows a client session in the Replication Server to wait for the switch to the new active database to complete.
wait for time	Specifies a time of day at which to unblock this command.

abort switch

Description

Aborts the switch active command, unless Replication Server has gone too far in the active switch process to abort it. The switch active command changes the active database in a warm standby application.

Syntax

abort switch for *logical_ds*.*logical_db*

Parameters

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

Examples

Example 1 Replication Server has gone too far in the active switch process to cancel. Wait for the switch to complete and enter another switch active command to return to the original active database.

```
abort switch for LDS.pubs2
```

```
Switch for logical connection LDS.pubs2 is beyond the
point where it can be aborted. Abort command fails.
```

Example 2 Replication Server has aborted the active switch. The active database has not changed.

```
abort switch for LDS.pubs2
```

```
Switch for logical connection LDS.pubs2 has been
aborted.
```

Usage

- The abort switch command attempts to cancel the switch active command.
- If there is no switch in progress for the logical connection, Replication Server returns an error message.
- If the command cancels the active switch successfully, you may have to restart the RepAgent for the active database.
- The switch active command cannot be cancelled after it reaches a certain point. If this is the case, you must wait for the switch active to complete. Then use switch active again to return to the original active database.

Permissions

abort switch requires "sa" permissions.

See also

switch active, admin logical_status, wait for switch

activate subscription

Description

For a subscription to a replication definition or a publication, starts the distribution of updates from the primary to the replicate database and sets the subscription status to ACTIVE. This command is part of the bulk materialization process, or part of the process of refreshing a publication subscription.

Syntax

```
activate subscription sub_name
for { table_rep_def | function_rep_def |
          publication pub_name | database replication definition db_repdef
          with primary at data_server.database }
 with replicate at data_server.database
[with suspension [at active replicate only]]
```

Parameters

sub_name
The name of the subscription to be activated.

for *table_rep_def*
Specifies the name of the table replication definition the subscription is for.

for *function_rep_def*
Specifies the name of the function replication definition the subscription is for.

for publication *pub_name*
Specifies the name of the publication the subscription is for.

for database replication definition *db_repdef*
Specifies the name of the database replication definition the subscription is for.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database. Use this clause only with a subscription for a publication.

with replicate at *data_server.database*
Specifies the location of the replicate data. If the replicate database is part of a warm standby application that uses logical connections, *data_server.database* is the name of the logical data server and database.

with suspension

Suspends the Data Server Interface (DSI) for the replicate database after changing the subscription status. While the DSI is suspended, Replication Server holds updates for the replicate database in a stable queue. After you load the initial data and resume the DSI, Replication Server applies the updates. In a warm standby application, this clause suspends the active database DSI and the standby DSI.

with suspension at active replicate only

In a warm standby application, suspends the active database DSI but not the standby DSI.

Examples

Example 1 Activates the subscription titles_sub for the table replication definition titles_rep, where the replicate database is SYDNEY_DS.pubs2. This command suspends the DSI.

```
activate subscription titles_sub
 for titles_rep
 with replicate at SYDNEY_DS.pubs2
 with suspension
```

Example 2 Activates the subscription myproc_sub for the function replication definition myproc_rep, where the replicate database is SYDNEY_DS.pubs2.

```
activate subscription myproc_sub
 for myproc_rep
 with replicate at SYDNEY_DS.pubs2
```

Example 3 Activates the subscription pubs2_sub for the publication pubs2_pub, where the primary database is TOKYO_DS.pubs2 and the replicate database is SYDNEY_DS.pubs2.

```
activate subscription pubs2_sub
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replicate at SYDNEY_DS.pubs2
```

Usage

- Use activate subscription to activate a subscription at the primary and replicate Replication Servers. The subscription can be to a table replication definition, function replication definition, or publication.
- This command begins the second step in the bulk materialization process. The first step is the creation of the subscription using define subscription.
- To complete bulk materialization, load the data from media, resume the connection to the replicate database if it was suspended, and execute validate subscription.

- Execute activate subscription at the Replication Server where you created the subscription.
- activate subscription changes the status of a subscription from DEFINED to ACTIVE. Subsequent updates at the primary data server are distributed through the primary Replication Server.
- If you have added any new articles to a publication with an existing subscription, you must refresh the publication subscription by materializing the new data in order to create subscriptions for the new articles.

  After using define subscription to begin this process, use activate subscription to activate the new article subscriptions. Then manually load the subscription data for the new article subscriptions, and use validate subscription to validate the publication subscription.
- When you activate a publication subscription, all of its article subscriptions are activated at the same time, rather than one at a time.
- This command modifies RSSD tables at multiple sites. Use check subscription at the primary and replicate Replication Servers to see the effects on each.
- For more information about subscription materialization, see the *Replication Server Administration Guide Volume 1*.

The *with suspension* clause

- When you use the with suspension clause, activate subscription suspends the DSI after changing the subscription status. This prevents the replicate Replication Server from sending updates for the replicated table before the subscription data is loaded.

  After the data is loaded at the replicate site, execute resume connection to apply the updates. If you do not use with suspension, you should prohibit updates to the primary version until after the subscription is materialized.
- If the database is part of a warm standby application, the with suspension clause suspends the DSI for the active database and standby DSI after changing the subscription status. This allows you to load the data into both databases before allowing updates to continue in the active database.

  If you load the data into the active database with logging (for example, by using logged bcp or by executing transactions in the active database), use the clause with suspension at active replicate only, so that the standby DSI is not suspended. In this case, you do not have to load the subscription data into the standby database because it is replicated from the active database.

Permissions	activate subscription can be executed by users with “create object” permission at the replicate Replication Server and “primary subscribe” permission at the primary Replication Server.
See also	check subscription, create subscription, define subscription, drop subscription, resume connection, validate subscription

add partition

Description
Makes a partition available to Replication Server. A partition can be a disk partition or an operating system file.

Note add partition and create partition are identical except for the command name. For backward compatibility, add partition is still supported as an alias for create partition but it will be depreciated in the future.

Syntax
For syntax information, see create partition.

Usage
For usage information, see create partition.

admin config

Description
Displays all Replication Server configuration parameters.

Syntax

```
admin config [,”connection” |, “logical_connection” |, “route”]
    [,server [,database]] [,configuration_name]
```

Parameters

”connection”
Displays connection configuration parameters.

”logical_connection”
Displays logical connection configuration parameters.

”route”
Displays route configuration parameters.

server [,*database*]
The data server and database being queried on.

If the configuration parameters to be displayed are related to a connection, the server must be a data server, and *database* must be supplied. If the parameters to be displayed are related to a route, server must be a Replication Server, and you cannot supply *database*.

configuration_name
The configuration parameter whose values and status you want to display.

Examples

Example 1 Displays all Replication Server global configuration parameters:

```
admin config
go
Configuration          Config   Run     Default
```

```
                      Value    Value   Value
-------------------   ------   -----   -------
cm_max_connections    65       65      64
dsi_cmd_batch_size    8193     8193    8192

Legal Values         Datatype   Status
-------------------  --------   -----------------
range: 1,2147483647  integer    Restart required
range: 1,2147483647  integer    Restart required
(2 rows affected)
```

Example 2 Displays all configuration parameters for route to Replication Server, TOKYO_RS:

```
admin config, "route", TOKYO_RS
```

Example 3 Displays all configuration parameters for connection to pdb1:

```
admin config,"connection",ost_wasatch_04,pdb1
go


Configuration         Config   Run     Default
                      Value    Value   Value
-------------------   ------   -----   -------
dsi_cmd_batch_size    NULL     NULL    8192

Legal Values         Datatype   Status
-------------------  --------   -----------------
range: 1,2147483647  integer    Connection/Route
                                restart required
(1 row affected)
```

Usage

Use admin config to retrieve the different types of configuration parameters—server, connection, logical connection, route—used to customize and tune the Replication Server:

For more information on configuring and tuning Replication Server parameters, refer to *Replication Server Administration Guide Volume 1 and Volume 2*.

admin disk_space

Description

Displays use of each disk partition accessed by the Replication Server.

Syntax

admin disk_space

Examples

Displays information about the disk partition:

```
admin disk_space

Partition     Logical          Part.Id
----------    -----------      -------
/dev/hdb2     partition_1      101

Total Segs    Used Segs        State
-----------   ---------        -------
         20           3        ON-LINE
```

Usage

Table 3-2 describes the output columns.

Table 3-2: Column descriptions for admin disk_space output

Column	Description
Partition	Device name used by the Replication Server
Logical	Logical name assigned to the partition
Part.Id	Partition ID
Total Segs	Total number of 1MB segments on a partition
Used Segs	Total segments currently in use by the Replication Server
State	State of this device. Can be: • ON-LINE – The device is normal • OFF-LINE – The device cannot be found • DROPPED – The device has been dropped but has not disappeared (some queues are using it)

Permissions

Any user may execute this command.

See also

admin who, alter partition, create partition, drop partition

admin echo

Description

Returns the string entered by the user.

Syntax

admin echo, *character_string* [, with_log]

Parameters

character_string
The character string entered by the user.

with_log
Writes the string entered by the user to the Replication Server log.

Examples

Example 1 The Replication Server returns “hello”, the character string entered by the user.

```
admin echo, hello

echo
-----
hello
```

Example 2 The Replication Server returns “Hello world!” and writes “Hello world!” to the Replication Server log.

```
admin echo, 'Hello world!', with_log
echo
------------
Hello world!
```

Usage

- Use admin echo to determine if the local Replication Server is running.
- This command does not function as a network echo. If you do not enter an argument, nothing is returned.

Permissions

Any user may execute this command.

admin get_generation

Description

Retrieves the generation number for a primary database.

Syntax

admin get_generation, *data_server*, *database*

Parameters

data_server
The data server with the primary database.

database
The database whose generation number you are retrieving.

Examples

```
admin get_generation, TOKYO_DS, pubs2
Current generation number for TOKYO_DS.pubs2 is 0
```

Usage

- The database generation number is the first 2 bytes of the origin queue ID generated by a RepAgent for log records. The generation number is a parameter of the Log Transfer Language (LTL) distribute command.

  For more information about the distribute command, refer to “distribute” in Chapter 5, “Creating a Replication Agent,” in the *Replication Server Design Guide*.

- The generation number should be incremented following a load for the primary database. Incrementing the number prevents Replication Server from ignoring (as duplicates) any transactions applied after the load.
- Increment the generation number by executing Adaptive Server dbcc settrunc in the Adaptive Server database.

Permissions

Any user may execute this command.

See also

dbcc settrunc

admin health

Description

Displays the status of the Replication Server.

Syntax

admin health

Examples

Displays the status of the Replication Server.

```
admin health
Mode         Quiesce     Status
-------      -------     -------
NORMAL       TRUE        HEALTHY
```

Usage

- Table 3-3 describes the output columns.

Table 3-3: Column descriptions for admin health output

Column	Description
Mode	The state of the Replication Server with regard to recovery. It is one of these values: • NORMAL – Replication Server is operating normally. • REBUILDING – This is a transient state while Replication Server executes the rebuild queues command. • RECOVERY – The Replication Server is in stand-alone mode and the rebuild queues command has been executed. • STANDALONE – Replication Server is not accepting or starting any connections. You can only enter this state by starting Replication Server with the -M flag. Exit from stand-alone mode by shutting down the Replication Server and restarting it without the -M flag.
Quiesce	Indicates if the Replication Server is quiesced. It is either: • TRUE – Replication Server is quiesced, that is, all messages have been flushed. • FALSE – Replication Server is not quiesced.
Status	Overall status of the Replication Server. It is either: HEALTHY – All threads are executing as expected. SUSPECT – A thread is down and the Replication Server expected it to be running. Or, a thread is in a “Connecting” state. The “Connecting” state means that either the server to which Replication Server is connecting is unavailable and a problem exists, or the Replication Server will connect successfully in a moment and the suspect status is transitory. You can see threads that are not running by executing admin who_is_down.

Permissions

Any user may execute this command.

See also

admin quiesce_check, admin quiesce_force_rsi, admin who, admin who_is_down, admin who_is_up, rebuild queues

admin log_name

Description	Displays the path to the current log file.
Syntax	admin log_name
Examples	Displays the path to the log file for the current Replication Server.

```
admin log_name
Log File Name
-----------------------------------
/work/log/TOKYO_RS.log
```

Usage	If you start Replication Server with the -e flag and give a full path name for the error log, admin log_name returns the full path. If you give a relative path name, admin log_name returns the relative path name in the Replication Server's current working directory.
Permissions	Any user may execute this command.
See also	admin set_log_name

admin logical_status

Description

Displays status information for logical connections.

Syntax

admin logical_status [, *logical_ds*, *logical_db*]

Parameters

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

Examples

This output shows the LDS.pubs2 logical connection in its normal, active state. The current active database is the pubs2 database in the TOKYO_DS data server. The standby database is the pubs2 database in the SYDNEY_DS data server. The TOKYO_RS Replication Server manages the logical connection. Both physical connections are active. No special operations are in progress.

```
admin logical_status, LDS, pubs2
```

```
Logical            Active              Active       Standby          Standby
Connection Name    Connection Name     Conn State   Connection       Conn
---------------    ---------------     ---------    Name             State
                                                    -----------      -----
[109] LDS.pubs2    [115]               Active/      [116]            Active/
                   TOKYO_DS.pubs2                   SYDNEY_DS.pubs2

Controller RS              Operation in    State of Operation in    Spid
---------------------      Progress        Progress                 -----
                           -------------   ----------------------
[16777317] TOKYO_RS        None            None
```

Usage

- Use admin logical_status to find the status of logical connections for an active database and a standby database in a warm standby application.
- If you do not specify *logical_ds* and *logical_db*, admin logical_status displays information about all logical connections controlled by this Replication Server.
- Table 3-4 describes the output columns.

Table 3-4: Column descriptions for admin logical_status output

Column	Description
Logical Connection Name	The DBID (database ID) for the logical connection and the logical data server and database names.
Active Connection Name	The DBID, the data server, and the database name for the current active database.

Column	Description
Active Connection State	A description of the status of the active connection. Can be active, suspended, or suspended by error.
Standby Connection Name	The DBID, the data server, and the database name for the current standby database.
Standby Connection State	A description of the status of the standby connection. Can be active, suspended, suspended by error, or waiting for marker.
Controller RS	The RSID (Replication Server ID) and name of the Replication Server that manages the logical, active, and standby databases.
Operation in Progress	A description of the operation in progress. Can be None, Switch Active, or Create Standby.
State of Operation in Progress	The current step in the operation.
Spid	The process ID for the server thread that is executing the operation.

Permissions

Any user may execute this command.

See also

abort switch, admin sqm_readers, admin who, create connection, create logical connection, switch active, wait for create standby, wait for switch

admin pid

Description	Displays the process ID of the Replication Server.
Syntax	admin pid
Examples	The process ID for the current Replication Server is 12032.

```
admin pid
pid
--------------
12032
```

Usage	Display the process ID of the Replication Server.
Permissions	Any user may execute this command.

admin quiesce_check

Description

Determines if the queues in the Replication Server have been quiesced.

Syntax

admin quiesce_check

Examples

Example 1 The TOKYO_RS Replication Server is quiescent.

```
admin quiesce_check
Replication Server TOKYO_RS is quiesced
```

Example 2 This message indicates that the system is not quiescent because there are unread messages in queue 103:1. The reported Read location (30.2) and Write location (32.1) show that more blocks in the queue have been written than read. Assuming no more blocks are written, the Read location must advance to segment 32, block 2, before the system becomes quiescent.

```
admin quiesce_check
Can't Quiesce. Queue 103:1 has not been read out.
 Write=32.1 Read=30.2
```

Usage

- admin quiesce_check determines if a Replication Server is quiescent.
- The Replication Server is quiescent if:
  - There are no subscription materialization queues.
  - Replication Server has read and processed all messages in all queues.
  - No inbound (RepAgent) queues contain undelivered committed transactions.
  - All messages in RSI queues have been sent to their destination Replication Servers and acknowledgments have been received.
  - All messages in DSI queues have been applied and acknowledgments received from data servers.

Permissions

Any user may execute this command.

See also

admin quiesce_force_rsi, suspend connection, suspend log transfer

admin quiesce_force_rsi

Description

Determines whether a Replication Server is quiescent and forces it to deliver and obtain acknowledgments for messages in RSI queues.

Syntax

admin quiesce_force_rsi

Examples

Example 1 The TOKYO_RS Replication Server is quiescent.

```
admin quiesce_force_rsi
Replication Server TOKYO_RS is quiesced
```

Example 2 This message indicates that the system is not quiescent because there are unread messages in queue 103:1. The reported Write location (32.1) and Read location (30.2) show that more blocks in the queue have been written than read.

```
admin quiesce_force_rsi
Can’t Quiesce. Queue 103:1 has not been read out.
Write=32.1 Read=30.2
```

Usage

- Execute suspend log transfer from all before you execute admin quiesce_force_rsi. This prevents RepAgents from connecting with the Replication Server.
- Execute this command after all inbound queues are quiescent.
- The Replication Server is quiescent if:
  - There are no subscription materialization queues
  - Replication Server has read all messages in all queues
  - No inbound (RepAgent) queues contain undelivered committed transactions
  - All messages in RSI queues have been sent to their destination Replication Servers and acknowledgments have been received
  - All messages in DSI queues have been applied and acknowledgments have been received from data servers
- RSI normally empties its queue every 30 seconds.

Permissions

Any user may execute this command.

See also

admin quiesce_check, suspend connection, suspend log transfer

admin rssd_name

Description

Displays the names of the data server and database for the RSSD.

Syntax

admin rssd_name

Examples

In the example, TOKYO_DS is the name of the data server, and TOKYO_RSSD is the name of the RSSD.

```
admin rssd_name
RSSD Dataserver    RSSD Database
---------------    ------------
TOKYO_DS           TOKYO_RSSD
```

Usage

Display the names of the data server and database for the RSSD.

Permissions

Any user may execute this command.

admin security_property

Description

Displays information about supported network-based security mechanisms and security services.

Syntax

admin security_property [, *mechanism_name*]

Parameters

mechanism_name
A supported network-based security mechanism.

Examples

```
admin security_property
Mechanism        Feature            Supported
---------------  ----------------   ------------
DCE              Unified Login      yes
DCE              Confidentiality    yes
DCE              Integrity          no
...
```

Usage

- When executed without options, displays the name of the default security mechanisms, the security services available for that mechanism, and whether available services are supported at your site.
- To execute admin security_property, network-based security must be enabled—use configure replication server to set the use_security_services parameter on—at the current Replication Server.

Permissions

Any user can execute this command.

See also

admin security_setting, alter connection, alter route, configure replication server, create connection, create route, set proxy

admin security_setting

Description

Displays network-based security parameters and values for the Replication Server.

Syntax

admin security_setting [, *rs_idserver* |, *rs_server* |, *data_server.database*]

Parameters

rs_idserver
The ID Server to which the current Replication Server connects.

rs_server
The Replication Server to which the current Replication Server connects.

data_server
The data server for the target database to which the current Replication Server connects.

database
The target database to which the current Replication Server connects.

Examples

```
admin security_setting
Server        Feature           Status
------------  --------------    -----------
Global        Unified Login     required
Global        Confidentiality   not_required
Global        Integrity         not_required
...
```

Usage

- To execute admin security_setting, network-based security must be enabled—use configure replication server to set the use_security_services parameter "on"— at the current Replication Server.
- If you execute admin security_setting without options, Replication Server displays default values configured with configure replication server.

Permissions

Any user may execute this command.

See also

admin security_property, alter connection, alter route, configure replication server, create connection, create route, set proxy

admin set_log_name

Description

Closes the existing Replication Server log file and opens a new log file.

Syntax

admin set_log_name, *log_file*

Parameters

log_file
The name of the new log file.

Examples

Opens a new log file called SYDNEY_RS.log. You can verify the path and log file name with the admin log_name command.

```
admin set_log_name,
  '/work/log/SYDNEY_RS.log'
```

Usage

- If this command fails, the original log file remains open.
- If the Replication Server is restarted, the log file name specified in the command line is used. If no name is specified in the command line, the default log file name is used.
- If you enter a log file name containing characters other than letters and numerals, enclose it in quotes. Do this, for example, if the log file name contains a period (.), as in the example below.
- admin set_log_name displays the name you enter. Enter an absolute path name to make the output most useful.

Permissions

Any user may execute this command.

See also

admin log_name

admin show_connections

Description

Displays information about all connections from the Replication Server to data servers and to other Replication Servers.

Syntax

admin show_connections

Examples

Displays connection data for this Replication Server.

```
                         admin show_connections
             Server            User              Database
             ------            ----              --------
             SYDNEY_DS         pubs2_maint       pubs2sb
             SYDNEY_RS         SYDNEY_RS_rsi     NULL

             State                  Owner              Spid
             -----                  -----              ----
             already_faded_out       DSI                 89
             active                  RSI                 53

       connection state   number  comments
       ----------------   -----   --------
       connecting         0       in the process of connecting to a server
       active             2       established connections owned and used by
                                  threads
       idle               0       established connections owned but not being
                                  used
       being_faded_out    0       idle connections that are being closed
       already_faded_out  0       idle connections that have been closed
       free               1       established connections not owned by any
                                  threads
       closed             61      closed connections not owned by any threads
       limbo              0       connection handles in state transition
       total              64      total number of connection handlers available
```

Usage

- This command displays information about database connections and routes from the current Replication Server.
- Table 3-5 describes the output from this command.

Table 3-5: Column descriptions for admin show_connections output

Column	Description
Server	The name of the data server or Replication Server to which this Replication Server is connected
User	The login name for this client
Database	The name of the database to which this Replication Server is connected (null for routes)

Column	Description
State	The state of this connection
Owner	Indicates the owner of the thread. One of these: DSI – Data Server Interface (to a database) RSI – Replication Server Interface (to a Replication Server)
Spid	Unique identifier for this thread
connection state	One of these values: • active – the connection is being used • already_faded_out – the connection is owned and closed • being_faded_out – the connection is owned and is being closed • closed – closed connections are not owned by any threads • connecting – connecting to a server • free – the connection is open and not owned by anyone • idle – the connection is owned but is not used • limbo – connection handles are in a state transition • total – the total number of connections
number	The number of connections of this type
comments	A description of the connection state field

Permissions

Any user may execute this command.

See also

alter connection, alter logical connection, alter route, create connection, create logical connection, create route, drop connection, drop logical connection, drop route, resume connection, suspend connection

admin show_function_classes

Description

Displays the names of existing function-string classes and their parent classes, and indicates the number of levels of inheritance.

Syntax

admin show_function_classes

Examples

```
admin show_function_classes

Class                          ParentClass                Level
--------                       ------------               -----
sql_derived_class              rs_default_function_class      1
DB2_derived_class              rs_db2_function_class          2
rs_db2_function_class          rs_default_function_class      1
rs_default_function_class      BASE_CLASS                     0
(and so on)
```

Usage

Level 0 is a base class such as rs_default_function_class, level 1 is a derived class that inherits from a base class, and so on.

Permissions

Any user may execute this command.

See also

alter connection, alter function string class, create connection, create function, create function string, create function string class, drop function string class, move primary

admin show_route_versions

Description

Displays the version number of routes that originate at the Replication Server and routes that terminate at the Replication Server.

Syntax

admin show_route_versions

Examples

In the example, the route version of ost_server_12.ost_server_11 is 11.5.0.

```
admin show_route_versions

Source RepServer  Dest. RepServer  Route Version
-----------------------------------------------------
ost_server_12     ost_server_11    1150
```

Usage

- The route version is the earliest site version of the source and destination Replication Server. If the route version is not the earliest site version, you need to upgrade the route.
- The version number determines which feature set in a mixed-version environment you can use with the route.
- For each route, admin show_route_versions displays the name of the source Replication Server, the name of the destination Replication Server, and the version of the route.

Permissions

Any user may execute this command.

See also

admin show_site_version, sysadmin fast_route_upgrade

admin show_site_version

Description

Displays the site version of the Replication Server.

Syntax

admin show_site_version

Examples

In the example, the Replication site version is 12.0.

```
admin show_site_version

Site Version
-------------
1200
```

Usage

Lets you use the software features in the corresponding release and prevents you from downgrading to an earlier release.

Permissions

Any user may execute this command.

See also

sysadmin site_version

admin sqm_readers

Description

Displays the read and delete points of the threads that are reading a stable queue.

Syntax

admin sqm_readers, *q_number*, *q_type*

Parameters

q_number

The ID number that Replication Server assigns to the queue. The number can be found in the output of the admin who, sqm command.

q_type

The type of the queue. Inbound queues have a type of 1. Outbound queues have a type of 0.

Examples

```
admin sqm_readers, 103, 1
```

```
RdrSpid     RdrType     Reader                          Index
-------     -------     ------                          -----
46          SQT         103:1 DIST LDS.pubs                 0
57          SQT         103:1 DSI 107 SYDNEY_DS.pubs2       1

First Seg.Block  Next Read  Last Seg.Block  Delete  WriteWait
---------------  ---------  --------------  ------  ---------
14.43            14.44      14.43                1          1
14.43            14.44      14.43                1          0
```

Usage

- admin sqm_readers reports the read point and the delete point for each Replication Server thread that is reading an inbound queue. You can use this information to help identify the cause when Replication Server fails to delete messages from queues.
- Replication Server cannot delete points beyond the minimum delete point of all threads that are reading the queue. The deletion point is the first segment block.
- Use the admin who, sqm command to find the *q_number*.
- Table 3-6 describes the output columns for the admin sqm_readers command.

Table 3-6: Column descriptions for admin sqm_readers output

Column	Description
RdrSpid	Unique identifier for this reader.
RdrType	The type of thread that is reading the queue.
Reader	Information about the reader. For a complete description of this information, see Table 3-7 on page 97.

Column	Description
Index	The index for this reader.
First Seg.Block	The first undeleted segment and block number in the queue. This information is useful when dumping queues.
Next Read	The next segment, block, and row to be read from the queue.
Last Seg.Block	The last segment and block written to the queue. This information is useful when dumping queues.
Delete	Whether or not the reader is allowed a delete. A value of “1” indicates that the reader is allowed a delete.
WriteWait	Whether or not the reader is waiting for a write. A value of “1” indicates that the reader is waiting for a write.

Permissions Any user may execute this command.

See also admin who, admin stats

admin stats

Description

Displays information and statistics about Replication Server operations.

Syntax

```
admin { stats | statistics } [, sysmon | "all"
          | module_name [, inbound | outbound ] [, display_name ] ]
          [, server[, database ] [instance_id  ]
          [, display |, save[, obs_interval ] ]
          [, sample_period ]
```

Parameters

sysmon
Displays statistics only for those counters identified as particularly important for performance and tuning purposes. Counters are selected from nearly all modules. This is the default.

"all"
Displays statistics from all counters.

module_name
Displays statistics from the named module's counters, where *module_name* is cm, dsi, dist, dsiexec, repagent, rsi, rsiuser, serv, sqm, sqt, sts, sync, and others. Use rs_helpcounter to obtain valid module names.

inbound | outbound
inbound and outbound are types of sqt or sqm. If neither inbound nor outbound is supplied for the sqt or sqm module, Replication Server reports statistics for both types of queues.

display_name
Is the name of a counter. Use rs_helpcounter to obtain valid display names. *display_name* is only used in conjunction with *module_name*.

server[, *database*]
If the statistics to be collected are related to a connection, *server* must be a dataserver and *database* must be supplied. If the statistics to be collected are related to a route, *server* must be a Replication Server and you are not allowed to supply *database*.

instance_id
Identifies a particular instance of a module, such as SQT or SQM, that has multiple instances. To view instance IDs, execute admin who and view the Info column.

The instance ID 0 indicates Replication Server-wide statistics. It is the instance ID of the Replication Server.

display
Displays statistics on the computer screen. This is the default.

save
: Saves statistics in the RSSD. Old sampling data is truncated or preserved, depending on the current setting of stats_reset_rssd.

obs_interval
: Specifies the length of each observation interval during the sampling period. If you do not specify an interval, there will be only one observation interval with a length equal to the sampling period. Each observation interval must be at least 15 seconds. Format can either be a numeric value in seconds, or “hh:mm[:ss]”.

sample_period
: Indicates the total sampling duration. The default value is zero, which reports the current counter values. With a non-zero value, the current counter values are reset and then collected for the specified sample period. Format can either be a numeric value in seconds, or “hh:mm[:ss]”.

Examples

Example 1 Collects outbound SQT statistics for connection 108 for two minutes and sends the data to the RSSD.

```
admin stats, sqt, outbound, 108, save, 120
```

Example 2 Collects outbound SQT statistics for connection 108 for two hours and sends data to the RSSD. In addition, the sample period is divided into observation intervals of 30 seconds each.

```
admin stats, sqt, outbound, 108, save, 30, "02:00:00"
```

Example 3 Displays statistics for the SQM and SQMR modules for the inbound queue for connection 102.

```
admin stats, sqm, inbound, 102
```

```
Report Time:            10/31/05 02:14:17 PM
Instance                        Instance ID  ModType/InstVal
--------------------------      -----------  ----------------
SQM, 102:1 pds01.tpcc                   102                 1

Monitor                 Obs     Last    Max     Avg ttl/obs
--------------------  ------  ------  ------  ------------
#*SegsActive               1       1       1             1

===========================================================================
Instance                        Instance ID  ModType/InstVal
------------------------------  -----------  ----------------
SQMR, 102:1 pds01.tpcc, 0 SQT           102                11

Observer                                  Obs    Rate x/sec
----------------------                  ------  ----------
```

```
SleepsWriteQ                                               4           0
```

Note In output, prefixes that precede counter names provide information about the counter. For example, a preceding # indicates a counter that is not reset, even if admin stats, reset is executed, and a preceding * indicates a counter that must be sampled, irrespective of the setting of stats_sampling. In this example, the SegsActive counter is always sampled and never reset.

Example 4 Collects statistics for the SleepsWriteQ counter in the SQM module.

```
admin stats, sqm, SleepsWriteQ
```

```
Report Time             10/31/05 02:17:03 PM

Instance                                Observer       Obs    Rate x/sec
-------------------------------------   -------------  ----   ----------
SQMR, 101:0 edsprs01.edbprs01, 0, DSI   SleepsWriteQ      0            0
SQMR, 102:0 pds01.tpcc, 0, DSI          SleepsWriteQ      0            0
SQMR, 102:1 pds01.tpcc, 0, DSI          SleepsWriteQ     20            0
SQMR, 103:0 rds01.tpcc, 0, DSI          SleepsWriteQ      0            0
```

Example 5 Starts sampling and saving statistics to the RSSD for one hour and thirty minutes at 20-second intervals:

```
admin stats, "all", save, 20, "01:30:00"
```

Usage

- By default, admin stats reports values for the sysmon counters.
- By default, admin stats does not report counters that show 0 (zero) observation. To change this behavior, set the stats_show_zero_counters configuration parameter on.
- If statistics are displayed on the computer screen, they are not stored in the RSSD. Similarly, if statistics are stored in the RSSD, they are not displayed on screen.
- If you use admin stats...*display_name* to display statistics for a particular counter, Replication Server always displays statistics for that counter, even if stats_sampling is off and the number of observations is zero.
- Use admin stats with the independent module name to collect statistics for dependent modules. You cannot collect statistics using the dependent module name in the admin stats command.

Independent module	**Dependent module**
Data Server Interface (DSI)	DSI Executor (DSIEXEC)
Stable Queue Manager (SQM)	SQM Reader (SQMR)
Thread Synchronization (SYNC)	SYNC Element (SYNCELE)

- admin stats displays different information for each type of counter:
  - Observers – values are reported for the number of observations and the number of observations per second.
  - Monitors – values are reported for the number of observations, last observed value, maximum observed value, and average value.
  - Counters – values are reported for the number of observations, the total of all observed values, the last observed value, the maximum observed value, the average value, and the number of observations per second.

Permissions Any user may execute this command.

See also configure replication server

admin stats, backlog

Description Reports the volume of replicated transactions awaiting distribution in the inbound and outbound queues in terms of segments and blocks.

Syntax admin { stats | statistics }, backlog

Examples Reports the transaction backlog for the inbound and outbound queues.

```
admin stats, backlog
```

```
Report Time:                 10/31/05 02:17:01 PM

Instance                      Monitor          Obs  Last  Max  Avg ttl/obs
----------------------------  ---------------  ---  ----  ---  -----------
SQMR 101:0 edsprs01.edbprs01,  *SQMRBacklogSeg   0     0    0            0
      0, DSI
SQMR 102:0 pds01.tpcc,         *SQMRBacklogSeg   0     0    0            0
      0, DSI
SQMR 102:1 pds01.tpcc,         *SQMRBacklogSeg 695     3    3            1
      0, SQT
SQMR 103:0 rds01.tpcc,         *SQMRBacklogSeg   0     0    0            0
      0, DSI
```

```
==========================================================================
Report Time:                10/31/05 02:56:11 PM

Instance                      Monitor            Obs Last Max Avg ttl/obs
----------------------------- ------------------ --- ---- --- -----------
SQMR 101:0 edsprs01.edbprs01, *SQMRBacklogBlock    0    0   0           0
      0, DSI
SQMR 102:0 pds01.tpcc,        *SQMRBacklogBlock    0    0   0           0
      0, DSI
SQMR 102:1 pds01.tpcc,        *SQMRBacklogBlock  692   50  64          29
      0, SQT
SQMR 103:0 rds01.tpcc,        *SQMRBacklogBlock  251    0   2           0
      0, DSI

==========================================================================
```

Usage

- admin stats, backlog collects data from the SQMRBacklogSeg and SQMRBacklogBlock counters.
- A segment is 1 MB and a block is 16K.

Permissions

Any user may execute this command.

admin stats, cancel

Description	Cancels the currently running asynchronous command. For multiple observation intervals, data already saved at the time of cancel is not deleted.
Syntax	admin { stats | statistics }, cancel
Usage	You can use admin stats, cancel to explicitly terminate the currently running asynchronous command. Replication Server does not allow other sampling commands when a sampling is already running in the background.
Permissions	Any user may execute admin stats, cancel.

admin stats, { md | mem | mem_in_use }

Description
: Reports information about memory usage.

Syntax
: admin { stats | statistics }, { md | mem | mem_in_use }

Parameters

md
: Reports Message Delivery statistics associated with the DIST and RSI users.

mem
: Reports current memory segment use according to segment size.

mem_in_use
: Reports current memory use in bytes.

Examples
: Reports total memory use in bytes.

```
admin stats, mem_in_use

Memory_in_Use
-------------
14215074
```

Usage
: Message Delivery statistics are associated with the DIST threads and RSI users.

Permissions
: Any user may execute this command.

admin stats, reset

Description	Resets all counters that can be reset.
Syntax	admin { stats | statistics }, reset
Examples	Resets all counters to zero. This command does not generate output.

```
admin stats, reset
```

Usage	You cannot use admin stats, reset or any other command to reset counters with the status bit 0X010 set in the status column of the rs_statcounters table.
Permissions	Any user may execute this command.
See also	admin stats, admin stats, status

admin stats, status

Description Displays configuration settings for monitors and counters.

Syntax admin { stats | statistics }, status

Examples

```
1> admin stats, status
2> go
```

```
Command in progress, sampling period 00:30:00, time
elapsed 00:02:32


Sybase Replication Server Statistics Configuration
================================================
Configuration              Default           Current
--------------------       ----------        --------
stats_sampling             off               on
stats_show_zero_counters   off               off
stats_reset_rssd            on                on
```

Usage

- Displays the default and current values of these configuration parameters:
  - stats_sampling – indicates whether sampling is on or off.
  - stats_show_zero_counters – specifies whether or not to display counters with zero observation since the last reset.

Permissions Any user may execute admin stats, status.

admin stats, { tps | cps | bps }

Description Reports the current throughput in terms of transactions, commands, or bytes per second.

Syntax admin { stats | statistics }, { tps | cps | bps }

Parameters

tps
Specifies that Replication Server reports the current throughput in transactions per second.

cps
Specifies that Replication Server reports the current throughput in commands per second.

bps

Specifies that Replication Server reports the current throughput in bytes per second.

Examples

Displays counters that calculate throughput in commands per second. Due to the length of the output, only a portion is shown here:

```
admin stats, cps
```

```
Report Time:            10/31/05 02:58:54 PM
Instance
Observer                             Obs    Rate x/sec
----------------------------------   -----  ----------
REP AGENT, pds01.tpcc *CmdsRecv      69876           0

(1 row affected)
=================================================================

Report Time:              10/31/05 02:58:54 PM
Instance
Observer                                    Obs    Rate x/sec
------------------------------------------  -----  -----------
SQM, 101:0 edsprs01.edbprs01 *CmdsWritten       0            0
SQM, 102:0 pds01.tpcc *CmdsWritten              0            0
SQM, 102:1 pds01.tpcc *CmdsWritten          69886           25
SQM, 103:0 rds01.tpcc *CmdsWritten          48174           17

(4 rows affected)
==========================================================================

Report Time:            10/31/05 02:58:54 PM
Instance
Observer                                         Obs         Rate x/sec
------------------------------------------------ ----------- ----------
SQMR, 101:0 edsprs01.edbprs01, 0, DSI *CmdsRead            0          0
SQMR, 102:0 pds01.tpcc, 0, DSI *CmdsRead                   0          0
SQMR, 102:1 pds01.tpcc, 0, SQT *CmdsRead               50499         18
SQMR, 103:0 rds01.tpcc, 0, DSI *CmdsRead               48144         17

(4 rows affected)
========================================================================

...
```

Usage

- When calculating throughput per second, Replication Server bases the calculation on the number of processed transactions and the number of elapsed seconds since the counters were last reset using admin stats, reset.
- Different modules report throughput for each type of calculation:
  - Transactions per second – are reported by the SQT, DIST, DSI, and other modules.
  - Commands per second – are reported by the RepAgent, RSIUSER, SQM, DIST, DSI, and RSI modules.
  - Bytes per second – are reported by the RepAgent, RSIUSER, SQM, DSI, and RSI modules. The SQM reports transactions in both bytes and blocks per second.

Permissions

Any user may execute this command.

admin time

Description Displays the current time of Replication Server.

Syntax admin time

Parameters None

Examples

```
admin time

Time
--------------------------
    Feb 15 2001 9:28PM
```

Usage

- admin time is useful for figuring out machine time, or time-zone differences while debugging or examining latency issues.
- This command is also useful in scripting, to figure out what time Replication Server initiates or completes tasks.

Permissions Any user may execute this command.

admin translate

Description

Performs a datatype translation on a value, displaying the results in delimited literal format.

Syntax

admin translate, *value*, *source_datatype*, *target_datatype*

Parameters

value
The literal representation of the value that is to be translated.

source_datatype
The name of a datatype (either a Replication Server native datatype or a datatype definition that describes the content and format of *value*).

target_datatype
The name of a datatype (either a Replication Server base datatype or a datatype definition that is the requested output for the translation).

Examples

Example 1 This examples translates the DB2 TIMESTAMP value '1999-06-22-14.35.23.123456' to the Oracle DATE value '22-Jan-99.'

```
admin translate, '1999-06-22-14.35.23.123456',
rs_db2_timestamp, rs_oracle_date
```

Example 2 This example translates the Adaptive Server binary value 0x1122aabb to the Oracle binary value '1122aabb.'

```
admin translate, 0x1122aabb, 'binary(4)',
        'rs_oracle_binary(4)'
```

Usage

- Delimit *value* according to the delimitation requirements of the base datatype of the source datatype.
- If *source_datatype* or *target_datatype* requires a length specification, for example char(255), enclose the datatype name in single quotes.
- The source and target datatypes may differ depending on whether you want to test class-level or column-level translations. Thus:
  - For class-level translations – use the published datatype for *source_datatype*.
  - For column-level datatypes – use the declared datatype for *source_datatype* and the published datatype for *target_datatype*.
- Use admin translate with the diagnostic version of Replication Server to trace errors in translations.

- For information about supported datatype translations, see the *Replication Server Heterogeneous Replication Guide*. For information about translating datatypes using heterogeneous datatype support (HDS), see the *Replication Server Administration Guide Volume 1*.

Permissions

Any user may execute this command.

See also

alter replication definition, create replication definition, alter connection, create connection

admin version

Description Displays the version number of the Replication Server software.

Syntax admin version

Examples

```
admin version

Version
-----------------------------------------------------
Replication Server/15.0/P/Sun_svr4/OS 5.8/1/OPT/Wed
Jan  4 17:47:58 2006 Copyright 1992, 2006
```

Usage

- The software version number of the Replication Server is the release level of the software product.
- The software version number does not, by itself, determine which capabilities you can use in the Replication Server. The system version number for the replication system and the site version number for the Replication Server also determine what features you can use.
- The Replication Server's site version number may be equal to or lower than the software version number. See sysadmin site_version for more information.
- The system version number for the replication system may be equal to or lower than the software version number. See sysadmin site_version for more information.

Permissions Any user may execute this command.

See also sysadmin site_version, sysadmin system_version

admin who

Description

Displays information about threads running in the Replication Server.

Syntax

admin who [, {dist | dsi | rsi | sqm | sqt}]

Parameters

dist

Returns information about Distributor threads. These threads distribute transactions in the inbound queue to replicate databases and Replication Servers.

dsi

Returns information about DSI threads. These threads apply replicated transactions to databases.

rsi

Returns information about RSI threads. These threads send messages to other Replication Servers.

sqm

Returns information about SQM threads. These threads manage Replication Server stable queues.

sqt

Returns information about SQT threads. These threads read queues and group functions into transactions.

Examples

Example 1 In the following example, admin who displays the state of all threads in the Replication Server. DSI scheduler threads are shown as "DSI" in the output. DSI executor threads are shown as "DSI EXEC." If the DSI is suspended when Replication Server starts up, the output shows only one DSI executor thread, even if more are configured.

```
admin who
```

```
Spid Name         State              Info
---- ----------   ------------------ -------------------------
 97  DIST        Active              103 LDS.pubs2
 98  SQT         Awaiting Wakeup     103:1  DIST LDS.pubs2
 68  SQM         Awaiting Message    103:0 LDS.pubs2
 89  DSI EXEC    Awaiting Message    106(1) SYDNEY_DS.pubs2sb
 91  DSI         Awaiting Command    106 SYDNEY_DS.pubs2sb
 21  DSI EXEC    Awaiting Message    101(1) TOKYO_DS.TOKYO_RSSD
 10  DSI         Awaiting Command    101 TOKYO_DS.TOKYO_RSSD
 16  DIST        Active              101 TOKYO_DS.TOKYO_RSSD
 17  SQT         Awaiting Wakeup     101:1  DIST TOKYO_DS.TOKYO_RSSD
 15  SQM         Awaiting Message    101:1 TOKYO_DS.TOKYO_RSSD
 14  SQM         Awaiting Message    101:0 TOKYO_DS.TOKYO_RSSD
 30  REP AGENT   Awaiting Command    TOKYO_DS.TOKYO_RSS
```

```
        USER
    4   DSI EXEC     Awaiting Message    104(1) TOKYO_DS.pubs2
    0   DSI          Awaiting Command    104 TOKYO_DS.pubs2
    8   REP AGENT    Awaiting Command    TOKYO_DS.pubs2
        USER
   53   RSI          Awaiting Wakeup     SYDNEY_RS
   52   SQM          Awaiting Message    16777318:0 SYDNEY_RS
        RSI USER     Inactive            TOKYO_RS
   11   dSUB         Active
    6   dCM          Awaiting Message
    9   dAIO         Awaiting Message
   12   dREC         Active              dREC
   71   USER         Active              sa
    5   dALARM       Awaiting Wakeup
   13   dSYSAM       Sleeping
```

Example 2 In the following example, the admin who, dist command displays information about each DIST thread in the Replication Server.

```
admin who, dist
```

```
Spid      State                       Info
-----     ----------------            ---------------------
21         Active                     102 SYDNEY_DS.SYDNEY_RSSD
22         Active                     106 SYDNEY_DS.pubs2

PrimarySite  Type   Status   PendingCmds   SqtBlocked
-----------  ----   ------   -----------   ----------
102          P      Normal             0            1
106          P      Normal             0            1

Duplicates   TransProcessed   CmdsProcessed   MaintUserCmds
----------   --------------   -------------   -------------
         0              715            1430               0
       290                1             293               0

NoRepdefCmds   CmdsIgnored   CmdMarkers
------------   -----------   ----------
           0             0            0
           0             0            1
```

Example 3 In this example, admin who, dsi displays information about each DSI scheduler thread running in the Replication Server.

```
admin who, dsi
```

```
 Spid                   State                          Info
```

```
  -----       ----------------        ----------------------
      8       Awaiting Message        101 TOKYO_DS.TOKYO_RSSD
     79       Awaiting Message        104 TOKYO_DS.pubs2
    145       Awaiting Message        105 SYDNEY_DS.pubs2sb

 Maintenance User  Xact_retry_times   Batch    Cmd_batch_size
  ---------------- ----------------   -----    --------------
  TOKYO_RSSD_maint                3      on              8192
       pubs2_maint                3      on              8192
       pubs2_maint                3      on              8192

 Xact_group_size        Dump_load     Max_cmds_to_log
  ---------------        ---------     ---------------
            65536              off                  -1
            65536              off                  -1
            65536              off                  -1

 Xacts_read       Xacts_ignored    Xacts_skipped
 ----------       -------------    -------------
         39                   0                0
          0                   0                0
       1294                   2                0


 Xacts_succeeded       Xacts_failed    Xacts_retried  Current Origin DB
 ---------------       ------------    -------------  -----------------
               0                 28                0                102
               0                  0                0                  0
               0                 93                0                104

 Current Origin QID     Subscription Name       Sub Command
 ------------------     -----------------       -----------
     0x000000000...                  NULL              NULL
     0x000000000...                  NULL              NULL
     0x000000000...                  NULL              NULL

 Current Secondary QID      Cmds_read      Cmds_parsed_by_sqt
 ---------------------      ---------      ------------------
                  NULL            129                       0
                  NULL              0                       0
                  NULL           6740                       0

 IgnoringStatus   Xacts_Sec_Ignored   GroupingStatus   TriggerStatus
 --------------   -----------------   --------------   -------------
       Applying                   0               on              on
       Applying                   0               on              on
       Applying                   0              off             off

 ReplStatus     NumThreads     NumLargeThreads     LargeThreshold
 ----------     ----------     ---------------     --------------
         on              1                   0                100
         on              1                   0                100
        off              3                   1                 20

 CacheSize  Serialization    Max_Xacts_in_group  Xacts_retried_blk
 ---------  ---------------  ------------------  -----------------
 0          wait_for_commit                  20                  0
 0          wait_for_commit                 200                  0
 0          wait_for_start                   20                  0

 CommitControl                  CommitMaxChecks CommitLogChecks
 ------------------------------ --------------- ---------------
 on                                         400             200
 on                                         400             200
 on                                         400             200

 CommitCheckIntvl IsolationLevel
 ---------------- ------------------------------
 1000             default
 1000             default
```

Example 4 In this example, admin who, rsi displays information about RSI threads.

```
admin who, rsi
```

```
 Spid     State             Info
 ----     ---------------   ----------------
   53     Awaiting Wakeup   SYDNEY_RS

 Packets Sent      Bytes Sent       Blocking Reads
 ------------      -------------    --------------
  3008.000000      624678.000000               269

 Locater Sent      Locater Deleted
 ------------      ---------------
 0x000000...       0x000000...
```

Example 5 In this example, admin who, sqm displays information about SQM threads.

```
admin who, sqm
```

```
 Spid      State                 Info
----      ----------------     -------------------------
  14      Awaiting Message     101:0 TOKYO_DS.TOKO_RSSD
  15      Awaiting Message     101:1 TOKYO_DS.TOKYO_RSSD
  52      Awaiting Message     16777318:0 SYDNEY_RS
  68      Awaiting Message     103:0 LDS.pubs2

Duplicates     Writes     Reads     Bytes
----------     ------     -----     -----
0                             0         0
0                          8867      9058
0               0.1        2037      2037
0               0.1.0         0         0



B Writes   B Filled  B Reads   B Cache   Save_Int:Seg
--------   -------   -------   -------   ------------
0           0                        0            0:0
0          34         44          2132           0:33
0           3         54           268            0:4
0           0         23             0       strict:O

First Seg.Block     Last Seg.Block      Next Read
---------------     --------------      ---------
        0.1               0.0              0.1.0
       33.10             33.10            33.11.0
        4.12              4.12             4.13.0
        0.1               0.0              0.1.0

Readers     Truncs
-------     ------
      1          1
      1          1
      1          1
      1          1
```

Example 6 In this example, admin who, sqt displays information about SQT threads.

```
admin who, sqt
```

```
Spid    State             Info
----    ---------------   -------------------------
  17    Awaiting Wakeup   101:1 TOKYO_DS.TOKYO_RSSD
  98    Awaiting Wakeup   103:1 DIST LDS.pubs2
  10    Awaiting Wakeup   101 TOKYO_DS.TOKYO_RSSD
```

```
   0    Awaiting Wakeup   106 SYDNEY_DSpubs2sb

Closed      Read      Open      Trunc
------      ----      ----      -----
     0      0         0         0
     0      0         0         0
     0      0         0         0
     0      0         0         0

Removed   Full   SQM Blocked    First Trans   Parsed
-------   ----   -----------    -----------   -------
      0      0             1              0         0
      0      0             1              0         0
      0      0             0              0         0
      0      0             0              0         0

SQM Reader   Change Oqids   Detect Orphans
----------   ------------   --------------
         0              0                0
         0              0                0
         0              0                1
         0              0                1
```

Usage

- If you use admin who with an option, you must include a comma before the option.
- To display information about all threads in the Replication Server, execute admin who with no options.

Output column descriptions for *admin who*

The spid, Name, State, and Info columns display when admin who is executed without options.The spid, State, and Info columns also display when any option is chosen.

spid **column**

This is a unique identifier for a thread running in the Replication Server. If a thread is suspended or down, this field is blank.

Name **and** ***Info*** **columns**

Name is the type of Replication Server thread. The contents of Info varies, depending upon the type of thread. Table 3-7 describes the Name and Info columns for each thread.

Table 3-7: Name and Info column for admin who output

Name	Description	Contents of info
dAlarm	Alarm daemon. This thread keeps track of alarms set by other threads, such as the fade-out time for connections and the subscription daemon retry interval.	Empty
dAIO	The asynchronous I/O daemon. It manages asynchronous I/O to stable queues for the Replication Server.	Empty
dCM	The daemon for the connection manager. It manages connections to data servers and other Replication Servers.	Empty
dREC	The recovery daemon. This thread sleeps for a configurable period of time (rec_daemon_sleep_time configuration parameter) and then initiates any recovery actions specified in the rs_recovery table.	Empty
dSUB	The subscription retry daemon. This thread wakes up after a configurable time-out period (sub_daemon_sleep_time configuration parameter) and attempts to restart any subscriptions that have failed.	Empty
dSYSAM	SySAM daemon. This thread keeps track of checked out licenses.	Empty
dVERSION	The version daemon. This thread activates briefly when the Replication Server is started for the first time after an upgrade. It communicates the Replication Server's new software version number to the ID Server.	The version of this Replication Server.
DIST	Distributor thread. Each primary database has a Distributor thread that reads transactions from the inbound queue, determines which subscriptions are interested, and forwards the transactions.	The names of the data server and database whose updates the thread is distributing.
DSI	DSI scheduler thread. This thread reads a stable queue via SQT and applies the transactions through the DSI Executor threads.	The name of the data server the thread writes to.
DSI EXEC	DSI executor thread. This thread executes the transactions on the replicate database and acts on errors that the replicate data server returns.	The ID of the DSI executor thread and the name of the data server it is connected to.
REP AGENT USER	A client connection that is a RepAgent thread. This thread verifies that RepAgent submissions are valid and writes them into the inbound queue.	The name of the primary data server and database whose log the RepAgent is forwarding.
RSI	RSI sender. This thread sends messages from one Replication Server to another.	The name of the Replication Server where messages are sent.

Name	Description	Contents of info
RSI User	Client connection thread for a Replication Server connected to this one. It writes messages destined for other Replication Servers or databases into outbound queues.	The name of the Replication Server connected to this one as a client.
RS User	Replication Server connection used to create or drop subscriptions at the primary Replication Server.	The name of the subscription owner.
SQM	Stable queue manager. This thread manages a Replication Server stable queue.	*Queue number:* An ID for a Replication Server or database. *Queue type:* 1 for the inbound queue, 0 for outbound queues. Any other number is the ID of a subscription the queue is for. *Queue identifier*: for these queues: • For queues used to spool messages to another Replication Server, it is the name of the other Replication Server. • For queues used to spool messages to databases, it is the name of the data server and database. • For queues used to spool messages related to a subscription being created or dropped, it is the name of the replication definition and the name of the subscription.
SQT	Stable queue transaction interface. This thread reads a stream of messages from a stable queue and reassembles the transactions in commit order. The Distributor and DSI use this thread.	Same as the corresponding SQM thread.
USER	Thread for a client executing RCL commands.	The login name of the client.

***State* column** The State column contains the thread execution status. Table 3-8 describes the valid states for Replication Server threads. The states for DSI threads are defined differently, depending on whether they are scheduler threads or executor threads. For the definitions, see the *Replication Server Troubleshooting Guide*.

Table 3-8: State column descriptions for admin who output

State	Description
Active	Actively processing a command.
Awaiting Command	The thread is waiting for a client to send a command.
Awaiting I/O	The thread is waiting for an I/O operation to finish.

State	Description
Awaiting Message	The thread is waiting for a message from an Open Server™ message queue.
Awaiting Wakeup	The thread has posted a sleep and is waiting to be awakened.
Connecting	The thread is connecting.
Down	The thread has not started or has terminated.
Getting Lock	The thread is waiting on a mutual exclusion lock.
Inactive	The status of an RSI User thread at the destination of a route when the source Replication Server is not connected to the destination Replication Server.
Initializing	The thread is being initialized.
Suspended	The thread has been suspended by the user.

Output column descriptions for *admin who, dist*

This command returns a table with a row for each DIST thread in the Replication Server. With the Spid, State, and Info columns, the table contains the columns shown in Table 3-9.

Table 3-9: Column descriptions for admin who, dist output

Column	Description
PrimarySite	The ID of the primary database for the SQT thread.
Type	The thread is a physical or logical connection.
Status	The thread has a status of “normal” or “ignoring.”
PendingCmds	The number of commands that are pending for the thread.
SqtBlocked	Whether or not the thread is waiting for the SQT.
Duplicates	The number of duplicate commands the thread has seen and dropped.
TransProcessed	The number of transactions that have been processed by the thread.
CmdsProcessed	The number of commands that have been processed by the thread.
MaintUserCmds	The number of commands belonging to the maintenance user.
NoRepdefCmds	The number of commands dropped because no corresponding table replication definitions were defined. In the case of Warm Standby, it is possible to have Rep Server create the replication definition. In multi-site availability (MSA), one defines database replication definitions. In either of these cases, if the replicated data originates from a source without a table replication definition, the counter is increased and replicated data proceeds to the target.
CmdsIgnored	The number of commands dropped before the status became “normal.”
CmdMarkers	The number of special markers that have been processed.

Output column descriptions for *admin who, dsi*

This command returns a table with a row for each running DSI scheduler thread in the Replication Server. If a DSI scheduler thread exists for a database but does not appear in the output of admin who, dsi, use resume connection to restart the data server interface for the database. Along with the Spid, State, and Info columns, the table contains the columns shown in Table 3-10.

Table 3-10: Column descriptions for admin who, dsi output

Column	Description
Maintenance User	The login name of the maintenance user applying the transactions.
Xact_retry_times	The number of times a failed transaction is retried if the error action is RETRY_LOG or RETRY_STOP.
Batch	Indicates if the batch option is on. If it is on, you can submit multiple commands as a batch to the data server.
Cmd_batch_size	The maximum size, in bytes, of a batch of output commands that you can send to the data server.
Xact_group_size	The maximum size, in bytes, of a transaction group consisting of source commands.
Dump_load	Indicates if the dump/load option is on. This configuration option coordinates dumps between primary and replicate databases.
Max_cmds_to_log	Maximum number of commands that can be logged into the exceptions log for a transaction. A value of -1 indicates an unlimited number of commands.
Xacts_read	The number of transactions read by the DSI from the outbound stable queue. This number should increase as the DSI applies transactions. You can use the information to monitor the rate of activity.
Xacts_ignored	The number of transactions determined to be duplicates. Typically, some transactions are ignored at start-up time because they were applied previously. Deletes from the DSI queue are delayed, so at start-up time, duplicates are detected and ignored. If you see a large number of ignored transactions, there is a chance that the rs_lastcommit table is corrupted. For more information, refer to the *Replication Server Troubleshooting Guide*.
Xacts_skipped	The number of transactions skipped by resuming the connection with skip first transaction.
Xacts_succeeded	The number of transactions applied successfully against the database.
Xacts_failed	The number of transactions that failed. Depending on the error mapping, some transactions may be written into the exceptions log. You should inspect the exceptions log.
Xacts_retried	The number of transactions that were retried.
Current Origin DB	The origin database ID for the current transaction.
Current Origin QID	If the state is Active, it is the Origin Queue ID of the begin log record of the transaction being processed. Otherwise, it is the Origin Queue ID of the begin log record of the last transaction processed.
Subscription Name	If the thread is processing a subscription, this is the name of the subscription.
Sub Command	If the thread is processing a subscription, this is the subscription command: activate, validate, drop, or unknown.

Column	Description
Current Secondary QID	If the thread is processing an atomic subscription applied incrementally, this column holds the queue ID of the current transaction.
Cmds_read	The number of commands read from the DSI queue.
Cmds_parsed_by_sqt	The number of commands parsed by SQT before being read by the DSI queue.
IgnoringStatus	Contains “Ignoring” if the DSI is ignoring transactions while waiting for a marker. Contains “Applying” if the DSI is executing transactions in the database.
Xacts_Sec_ignored	In a warm standby application, the number of transactions that were ignored after the switchover.
GroupingStatus	Contains “on” if the DSI is executing transactions in groups. Contains “off” if the DSI is executing transactions one at a time.
TriggerStatus	Contains “on” if set triggers is on. Contains “off” if set triggers is off.
ReplStatus	Indicates whether the Replication Server replicates transactions in the database. The default is “off” for standby databases. The default is “on” for all other databases.
NumThreads	The number of parallel DSI threads in use.
NumLargeThreads	The number of parallel DSI threads reserved for use with large transactions.
LargeThreshold	In a parallel DSI configuration, the number of commands allowed in a transaction before it is considered large.
CacheSize	The maximum SQT cache memory for the database connection, in bytes. The default, 0, means that the current setting of the sqt_max_cache_size parameter is used as the maximum SQT cache memory.
Serialization	The method used to maintain serial consistency when parallel DSI threads are used.
Max_Xacts_in_group	The maximum number of transactions in a group. The default is 20. You can configure this number using the alter connection command.
Xacts_retried_blk	The number of times the DSI rolled back a transaction due to exceeding maximum number of checks for lock contention.
CommitControl	Indicates if commit control is internal or external. Set to true if internal.
CommitMaxChecks	Indicates the maximum number of lock contention attempts before rolling back transaction and retrying.
CommitLogChecks	Indicates the maximum number of lock contention attempts before logging a message.
CommitCheckIntvl	Amount of time, in milliseconds, a transaction waits before issuing a check for lock contention.
IsolationLevel	Database isolation level for DSI connection.

Output column descriptions for *admin who, rsi*

This command displays information about RSI threads that send messages to other Replication Servers. Along with the Spid, State, and Info columns, admin who, rsi contains the columns shown in Table 3-11.

Table 3-11: Column descriptions for admin who, rsi output

Column	Description
Packets Sent	The number of network packets sent.
Bytes Sent	The total number of bytes sent.
Blocking Reads	The number of times the stable queue was read with a blocking read.
Locater Sent	The locator of the last message sent (contains the queue segment, block and row).
Locater Deleted	The last locator that the recipient acknowledged and that has been deleted by Replication Server.

Output column descriptions for *admin who, sqm*

This command displays information about SQM threads that manage Replication Server stable queues. Along with the Spid, State, and Info columns, admin who, sqm contains the columns shown in Table 3-12.

Table 3-12: Column descriptions for admin who, sqm output

Column	Description
Duplicates	The number of duplicate messages detected and ignored. There are usually some duplicate messages at start-up.
Writes	The number of messages written into the queue.
Read	The number of messages read from the queue. This usually exceeds the number of writes because the last segment is read at start-up to determine where writing is to begin. Also, long transactions may cause messages to be reread.
Bytes	The number of bytes written.
B Writes	The number of 16K blocks written. It may be greater than *Bytes*/16K because not every 16K block written is full. You can determine the density of blocks by dividing *Bytes* by *B Writes*.
B Filled	The number of 16K blocks written to disk because they are filled.
B Reads	The number of 16K blocks read.
B Cache	The number of 16K blocks read that are in cache.
Save_Int:Seg	The Save_Int interval and the oldest segment in the Save_Int list. The Save_Int interval is the number of minutes the Replication Server maintains an SQM segment after all messages in the segment have been acknowledged by targets. For example, a value of 5:88 indicates a Save_Int interval of 5 minutes, where segment 88 is the oldest segment in the Save_Int list. This feature provides redundancy in the event of replication system failure. For example, a Replication Server could lose its disk partitions while receiving data from another Replication Server. The Save_Int feature lets the sending Replication Server re-create all messages saved during the Save_Int interval. A *Save_Int* value of "strict" may be used when a queue is read by more than one reader thread. Replication Server maintains the SQM segment until all threads reading the queue have read the messages on the segment and applied them to their destination.

Column	Description
First Seg.Block	The first undeleted segment and block number in the queue. If the figures for First Seg.Block and Last Seg.Block do not match, data remains in the queue for processing. This information is useful when dumping queues. For more information, refer to the *Replication Server Troubleshooting Guide*.
Last Seg.Block	The last segment and block written to the queue. If the figures for First Seg.Block and Last Seg.Block do not match, data remains in the queue for processing. This information is useful when dumping queues. For more information, refer to the *Replication Server Troubleshooting Guide*.
Next Read	The next segment, block, and row to be read from the queue.
Readers	The number of threads that are reading the queue.
Truncs	The number of truncation points for the queue.

Output column descriptions for *admin who, sqt*

SQT threads read transactions from a stable queue and pass them to the SQT reader in commit order. The reader can be a DIST or a DSI thread.

SQT stores the transactions it is processing in a memory cache. The Closed, Read, Open, Trunc, and Removed columns shown in Table 3-13 apply to transactions in the SQT cache.

Table 3-13: Column descriptions for admin who, sqt output

Column	Description
Closed	The number of committed transactions in the SQT cache. The transactions have been read from the stable queue and await processing.
Read	The number of transactions processed, but not yet deleted from the queue.
Open	The number of uncommitted or unaborted transactions in the SQT cache.
Trunc	The number of transactions in the transaction cache. Trunc is the sum of the Closed, Read, and Open columns.
Removed	The number of transactions whose constituent messages have been removed from memory. This happens when the SQT processes large transactions. The messages are reread from the stable queue.
Full	Indicates that the SQT has exhausted the memory in its cache. This is not a problem as long as there are closed or read transactions still awaiting processing. If the SQT cache is often full, consider raising its configured size. To do this, see alter connection.
SQM Blocked	1 if the SQT is waiting on SQM to read a message. This state should be transitory unless there are no closed transactions.

Column	**Description**
First Trans	This column contains information about the first transaction in the queue and can be used to determine if it is an unterminated transaction. The column has three pieces of information: • ST: Followed by O (open), C (closed), R (read), or D (deleted) • Cmds: Followed by the number of commands in the first transaction • qid: Followed by the segment, block, and row of the first transaction
Parsed	The number of transactions that have been parsed.
SQM Reader	The index of the SQM reader handle.
Change Oqids	Indicates that the origin queue ID has changed.
Detect Orphans	Indicates that it is doing orphan detection.

Permissions

Any user may execute this command.

admin who_is_down

Description
Displays information about Replication Server threads that are not running.

Syntax
admin who_is_down

Examples

```
admin who_is_down

Spid   Name             State         Info
----   --------------   -----------   ------------
       RSI              Suspended     SYDNEY_RS
```

Usage

- The Spid column in the output of admin who_is_down is always empty. There are no processes for threads that are not running.
- Execute admin who_is_down when admin health shows that the Replication Server is suspect. The output for this command does not list threads that are in a state of "Connecting," which could be the cause of the suspect health.
- For a description of the output from this command, see admin who.

Permissions
Any user may execute this command.

See also
admin health, admin who, admin who_is_up

admin who_is_up

Description Displays information about Replication Server threads that are running.

Syntax admin who_is_up

Examples

```
admin who_is_up
```

```
 Spid  Name       State            Info
  ----  --------   ---------------   --------------------
    97  DIST       Active            103 LDS.pubs2
    98  SQT        Awaiting Wakeup   103:1 DIST LDS.pubs2
    96  SQM        Awaiting Message  103:1 LDS.pubs2
    68  SQM        Awaiting Message  103:0 LDS.pubs2
    89  DSI EXEC   Awaiting Message  106(1) SYDNEY_DS.pubs2sb
    91  DSI        Awaiting Command  106 SYDNEY_DS.pubs2sb
    21  DSI EXEC   Awaiting Message  101(1) TOKYO_DS.TOKYO_RSSD
    10  DSI        Awaiting Command  101 TOKYO_DS.TOKYO_RSSD
    16  DIST       Active            101 TOKYO_DS.TOKYO_RSSD
    17  SQT        Active            101:1 DIST TOKYO_DS.TOKYO
    15  SQM        Awaiting Message  101:1 TOKYO_DS.TOKYO_RSSD
    14  SQM        Awaiting Message  103:1 TOKYO_DS.TOKYO_RSSD
    30  REP AGENT  Awaiting Command  TOKYO_DS.TOKYO_RSSD
        USER
     4  DSI EXEC   Awaiting Message  104(1) TOKYO_DS.pubs2
     9  dAIO       Awaiting Message
    12  dREC       Active            dREC
    61  USER       Active            sa
     5  dALARM     Awaiting Wakeup
```

Usage

- For a description of the output, see admin who.

Permissions Any user may execute this command.

See also admin who, admin who_is_down

allow connections

Description	Places Replication Server in recovery mode for specified databases.
Syntax	allow connections

Usage

- Execute allow connections to begin replaying log records from reloaded dumps.
- Start Replication Server in stand-alone mode and execute set log recovery for each database whose log you are replaying.
- After executing allow connections, Replication Server accepts connect requests only from RepAgents started in recovery mode for the specified databases. This ensures that Replication Server receives the replayed log records before current transactions.
- If you restart Replication Server in stand-alone mode and execute allow connections without first executing set log recovery commands, Replication Server moves from stand-alone mode to normal mode.
- For detailed recovery procedures, see the *Replication Server Administration Guide Volume 2*.

Permissions	allow connections requires “sa” permission.
See also	ignore loss, rebuild queues, set log recovery

alter connection

Description

Changes the attributes of a database connection.

Syntax

```
alter connection to data_server.database {
        set function string class [to] function_class |
        set error class [to] error_class |
        set password [to] passwd |
        set log transfer [to] {on | off} |
        set database_param [to] 'value' |
        set security_param [to] 'value' |
        set security_services [to] 'default']}
```

Parameters

data_server
The data server that holds the database whose connection is to be altered.

database
The database whose connection is to be altered.

function_class
The function-string class to use with the data server.

Note If you already have a DB2 database configured as a replicate database with an earlier version of Replication Server, continue to use the earlier version with Replication Server 12.0 and later and its HDS feature. The 12.0 and later function strings may not be compatible with earlier function string versions.

error_class
The error class to use to handle database errors.

passwd
The new password to use with the login name for the database connection. You must specify a password if network-based security is not enabled.

log transfer on
Allows the connection to send transactions from a RepAgent to the Replication Server.

log transfer off
Stops the connection from sending transactions from a primary database RepAgent.

database_param
The parameter that affects database connections from the Replication Server.

value
A character string containing a new value for the option.

Note Parameters and values are described in Table 3-14.

Table 3-14: Parameters affecting database connections

database_param	value
batch	Specifies how Replication Server sends commands to data servers. When batch is "on," Replication Server may send multiple commands to the data server as a single command batch. When batch is "off," Replication Server sends commands to the data server one at a time. Default: on
batch_begin	Indicates whether a begin transaction can be sent in the same batch as other commands (such as insert, delete, and so on). Default: on
command_retry	The number of times to retry a failed transaction. The value must be greater than or equal to 0. Default: 3
disk_affinity	Specifies an allocation hint for assigning the next partition. Enter the logical name of the partition to which the next segment should be allocated when the current partition is full. Default: off
dist_stop_unsupported_cmd	When dist_stop_unsupported_cmd is on, DIST suspends itself if a command is not supported by downstream Replication Server. If it is off, DIST ignores the unsupported command. Regardless of dist_stop_unsupported_cmd parameter's setting, Replication Server always logs an error message when it sees the first instance of a command that cannot be sent over to a lower-version Replication Server. Default: off
db_packet_size	The maximum size of a network packet. During database communication, the network packet value must be within the range accepted by the database. Default: 512-byte network packet for all Adaptive Server databases Maximum: 16,384 bytes

database_param	**value**
dsi_charset_convert	The specification for handling character-set conversion on data and identifiers between the primary Replication Server and the replicate Replication Server. This parameter applies to all data and identifiers to be applied at the DSI in question. The values are: • "on" – convert from the primary Replication Server character set to the replicate Replication Server character set; if character sets are incompatible, shut down the DSI with an error. • "allow" – convert where character sets are compatible; apply any unconverted updates to the database, as well. • "off" – do not attempt conversion. This option is useful if you have different but compatible character sets and do not want any conversion to take place. During subscription materialization, a setting of "off" behaves as if it were "allow." Default: on
dsi_cmd_batch_size	The maximum number of bytes that Replication Server places into a command batch. Default: 8192 bytes
dsi_cmd_separator	The character that separates commands in a command batch. Default: newline (\n) **Note** You must update this parameter in an interactive mode, not by executing a DDL-generated script, or any other script. You cannot reset dsi_cmd_separator by running a script.
dsi_commit_check_locks_intrvl	The number of milliseconds (ms) the DSI executor thread waits between executions of the rs_dsi_check_thread_lock function string. Used with parallel DSI. Default: 1000ms (1 second) Minimum: 0 Maximum: 86,400,000 ms (24 hours)
dsi_commit_check_locks_logs	The number of times the DSI executor thread executes the rs_dsi_check_thread_lock function string before logging a warning message. Used with parallel DSI. Default: 200 Minimum: 1 Maximum: 1,000,000

database_param	value
dsi_commit_check_locks_max	The maximum number of times a DSI executor thread checks whether it is blocking other transactions in the replicate database before rolling back its transaction and retrying it. Used with parallel DSI. Default: 400 Minimum: 1 Maximum: 1,000,000
dsi_commit_control	Specifies whether commit control processing is handled internally by Replication Server using internal tables (on) or externally using the rs_threads system table (off). Default: on
dsi_exec_request_sproc	Turns on or off request stored procedures at the DSI of the primary Replication Server. Default: on
dsi_fadeout_time	The number of seconds of idle time before a DSI connection is closed. A value of “-1” indicates that a connection will not close. Default: 600 seconds
dsi_ignore_underscore_name	When the transaction partitioning rule is set to “name,” specifies whether or not Replication Server ignores transaction names that begin with an underscore. Values are “on” and “off.” Default: on
dsi_isolation_level	Specifies the isolation level for transactions. The ANSI standard and Adaptive Server supported values are: • 0 – ensures that data written by one transaction represents the actual data. • 1 – prevents dirty reads and ensures that data written by one transaction represents the actual data. • 2 – prevents nonrepeatable reads and dirty reads, and ensures that data written by one transaction represents the actual data. • 3 – prevents phantom rows, nonrepeatable reads, and dirty reads, and ensures that data written by one transaction represents the actual data. **Note** Data servers supporting other isolation levels are supported as well through the use of the rs_set_isolation_level function string. Replication Server supports all values for replicate data servers. The default value is the current transaction isolation level for the target data server.

database_param	value
dsi_keep_triggers	Specifies whether triggers should fire for replicated transactions in the database. Set off to cause Replication Server to set triggers off in the Adaptive Server database, so that triggers do not fire when transactions are executed on the connection. Set on for all databases except standby databases. Default: on (except standby databases)
dsi_large_xact_size	The number of commands allowed in a transaction before the transaction is considered to be large. Minimum: 4 Default: 100
dsi_max_cmds_to_log	The number of commands to write into the exceptions log for a transaction. Default: -1 (all commands)
dsi_max_xacts_in_group	Specifies the maximum number of transactions in a group. Larger numbers may improve data latency at the replicate database. Range of values: 1 – 100. Default: 20
dsi_max_text_to_log	The number of bytes to write into the exceptions log for each rs_writetext function in a failed transaction. Change this parameter to prevent transactions with large text, unitext, image or rawobject columns from filling the RSSD or its log. Default: -1 (all text, unitext, image, or rawobject columns)
dsi_num_large_xact_threads	The number of parallel DSI threads to be reserved for use with large transactions. The maximum value is one less than the value of dsi_num_threads. Default: 0
dsi_num_threads	The number of parallel DSI threads to be used. The maximum value is 255. Default: 1
dsi_partitioning_rule	Specifies the partitioning rules (one or more) the DSI uses to partition transactions among available parallel DSI threads. Values are origin, ignore_origin, origin_sessid, time, user, name, and none. See the *Replication Server Administration Guide Volume 2* for detailed information. Default: none

database_param	value
dsi_replication	Specifies whether or not transactions applied by the DSI are marked in the transaction log as being replicated. When dsi_replication is set to "off," the DSI executes set replication off in the Adaptive Server database, preventing Adaptive Server from adding replication information to log records for transactions that the DSI executes. Since these transactions are executed by the maintenance user and, therefore, not usually replicated further (except if there is a standby database), setting this parameter to "off" avoids writing unnecessary information into the transaction log. dsi_replication must be set to "on" for the active database in a warm standby application for a replicate database, and for applications that use the replicated consolidated replicate application model. Default: on ("off" for standby database in a warm standby application)
dsi_replication_ddl	Supports bidirectional replication by specifying whether or not transactions are to be replicated back to the original database. When dsi_replication_ddl is set to on, DSI sends set replication off to the replicate database, which instructs it to mark the succeeding DDL transactions available in the system log not to be replicated. Therefore, these DDL transactions are not replicated back to the original database, which enables DDL transaction replication in bidirectional MSA replication environment. Default: off
dsi_rs_ticket_report	Determines whether to call function string rs_ticket_report or not. rs_ticket_report function string is invoked when dsi_rs_ticket_report is set to on. Default: off
dsi_serialization_method	Specifies the method used to determine when a transaction can start, while still maintaining consistency. In all cases, commit order is preserved. These option methods are ordered from most to least amount of parallelism. Greater parallelism can lead to more contention between parallel transactions as they are applied to the replicate database. To reduce contention, use the dsi_partition_rule option. • no_wait – specifies that a transaction can start as soon as it is ready—without regard to the state of other transactions. • wait_for_start – specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started. • wait_for_commit – specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it is ready to commit.

database_param	value
	These options are retained only for backward compatibility: • none – same as wait_for_start. • single_transaction_per_origin – same as wait_for_start with dsi_partitioning_rule set to origin. **Note** The isolation_level_3 value is no longer supported as a serialization method but it is the same as setting dsi_serialization_method to wait_for_start and dsi_isolation_level to 3. Default: wait_for_commit
dsi_sql_data_style	Formats datatypes (particularly date/time, binary, bit and money) to be compatible with: • DB2 (“db2”) • Lotus Notes (“notes”), or • SQL Anywhere, formerly Watcom SQL (“watcom”), or • SQL Remote (“sqlremote”) To support Transact-SQL instead, set this parameter to any value other than those listed above. When you configure a connection to DB2, specify the name of the NetGateway using the data_server parameter in the main clause of alter connection. When you configure a connection to Lotus Notes, SQL Anywhere, or any other ODBC data source, specify the connection as *replication_driver_name*.*odbc_data_source_name*. Refer to the *ODBC Driver Reference Guide*, which is part of the Open Client/Server™ version 11.1.1 collection, for more information. **Note** Obsolete for Replication Server version 12.0 and later. Retained for compatibility with older Replication Servers. Default: “ ” (for Adaptive Server)
dsi_sqt_max_cache_size	Maximum SQT (Stable Queue Transaction interface) cache memory for the database connection, in bytes. The default, “0,” means that the current setting of sqt_max_cache_size is used as the maximum cache size for the connection. Default: 0

database_param	value
dsi_xact_group_size	The maximum number of bytes, including stable queue overhead, to place into one grouped transaction. A grouped transaction is multiple transactions that the DSI applies as a single transaction. A value of –1 means no grouping. Sybase recommends that you set dsi_xact_group_size to the maximum value and use dsi_max_xacts_in_group to control the number of transactions in a group. **Note** Obsolete for Replication Server version 15.0 and later. Retained for compatibility with older Replication Servers. Maximum: 2,147,483,647 Default: 65,536 bytes
dsi_text_convert_multiplier	Changes the length of text or unitext datatype columns at the replicate site. Use dsi_text_convert_multiplier when text or unitext datatype columns must expand or contract due to character set conversion. Replication Server multiplies the length of primary text or unitext data by the value of dsi_text_convert_multiplier to determine the length of text or unitext data at the replicate site. Its type is float. • If the character set conversion involves expanding text or unitext datatype columns, set dsi_text_convert_multiplier equal to or greater than 1.0. • If the character set conversion involves contracting text or unitext datatype columns, set dsi_text_convert_multiplier equal to or less than 1.0. Default: 1
dump_load	Set to “on” at replicate sites only to enable coordinated dump. See the *Replication Server Administration Guide Volume 2* for details. Default: off
dynamic_sql	Turns dynamic SQL feature on or off for a connection. Other dynamic SQL related configuration parameters will take effect only if this parameter is set to on. Default: off
dynamic_sql_cache_management	Manages the dynamic SQL cache for a connection. Values: • mru (default) – specifies that once dynamic_sql_cache_size is reached, the old dynamic SQL prepared statements are deallocated to give room for new statements. • fixed – specifies that once the dynamic_sql_cache_size is reached, allocation for new dynamic SQL statements stops.

database_param	value
dynamic_sql_cache_size	Allows Replication Server to estimate how many database objects can use dynamic SQL for a connection. You can use dynamic_sqll_cache_size to limit resource demand on a data server. Default: 20 Minimum: 1 Maximum: 65,535
exec_cmds_per_timeslice	Specifies the number of LTL commands an LTI or RepAgent Executor thread can possess before it must yield the CPU to other threads. Default: 5 Minimum: 1 Maximum: 2,147,483,648
exec_sqm_write_request_limit	Specifies the amount of memory available to the LTI or RepAgent Executor thread for messages waiting to be written to the inbound queue. Default: 1MB Minimum: 16KB Maximum: 2GB
md_sqm_write_request_limit	Specifies the amount of memory available to the Distributor for messages waiting to be written to the outbound queue. **Note** In Replication Server 12.1, md_sqm_write_request_limit replaces md_source_memory_pool. md_source_memory_pool is retained for compatibility with older Replication Servers. Default: 1MB Minimum: 16KB Maximum: 2GB
parallel_dsi	Provides a shorthand method for configuring parallel DSI threads. A setting of “on” configures these values: • dsi_num_threads to 5 • dsi_num_large_xact_threads to 2 • dsi_serialization_method to “wait_for_commit” • dsi_sqt_max_cache_size to 1 million bytes. A setting of “off” configures these parallel DSI values to their defaults. You can set this parameter to “on” and then set individual parallel DSI configuration parameters to fine-tune your configuration. Default: off
save_interval	The number of minutes that the Replication Server saves messages after they have been successfully passed to the destination data server. See the *Replication Server Administration Guide Volume 2* for details. Default: 0 minutes

database_param	value
sub_sqm_write_request_limit	Specifies the memory available to the subscription materialization or dematerialization thread for messages waiting to be written to the outbound queue. Default: 1MB Minimum: 16KB Maximum: 2GB
use_batch_markers	Controls the processing of function strings rs_batch_start and rs_batch_end. If use_batch_markers is set to on, the rs_batch_start function string is prepended to each batch of commands and the rs_batch_end function string is appended to each batch of commands. Set use_batch_markers to on only for replicate data servers that require additional SQL to be sent at the beginning or end of a batch of commands that is not contained in the rs_begin function string. Default: off

security_param
A parameter that affects network-based security for connections. See Table 3-27 on page 240 for a list of parameters and a description of values.

set security_services to 'default'
Resets all network-based security features for the connection to match the global settings of your Replication Server.

Examples

Example 1 Changes the function-string class for the pubs2 database in the TOKYO_DS data server to sql_derived_class:

```
suspend connection to TOKYO_DS.pubs2

alter connection to TOKYO_DS.pubs2b
set function string class to sql_derived_class

resume connection to TOKYO_DS.pubs2
```

Example 2 Changes the number of LTL commands the LTI or RepAgent Executor thread can process before it must yield the CPU to other threads:

```
suspend connection to TOKYO_DS.pubs2
alter connection to TOKYO_DS.pubs2b
set exec_cmds_per_timeslice to ’10’
resume connection to TOKYO_DS.pubs2
```

Usage

- Use suspend connection to suspend activity on the connection before altering it.

- Execute alter connection at the Replication Server where the connection was created.
- Before you use log transfer off to stop data transfer from a primary database, be sure there are no replication definitions defined for data in the database.
- To change the route to a Replication Server, use alter route.
- Use set function string class [to] *function_class* to activate class-level translations for non-Sybase data servers.
- You can set connection parameters using the alter connection parameter.
- Execute alter connection at the Replication Server where the connection was created.

Database connection parameters

- Use alter connection to change the configuration parameters of a DSI or a database connection. To change a DSI configuration value, suspend the connection to the DSI, change the value, and then resume the connection to the DSI. This procedure causes the new value to take effect.
- Replication Server configuration parameters are stored in the rs_config system table. Some parameters can be modified by updating rows in the table. See the *Replication Server Administration Guide Volume 1* for more information.
- See the *Replication Server Administration Guide Volume 2* for more information about configuring parallel DSI threads.
- Use assign action to enable retry of transactions that fail due to specific data server errors.
- Before you change the function-string class, make sure that the class and all the required function strings exist for the new class.
- Before you change the error class, make sure the new class exists.
- Change the character for data servers that require a command separator to recognize the end of a command.

  If you have specified a different separator character and want to change it back to a newline character, enter the *alter connection* command as follows:

  ```
  alter connection to data_server.database
    set  to '<Return>'
  ```

where you press the Return key, and no other characters, between the two single-quote characters.

The *dsi_partitioning_rule* parameter

You can specify more than one partitioning rule at a time. Separate values with a comma, but no spaces. For example:

```
alter connection to data_server.database
  set dsi_partitioning_rule to ‘origin,time’
```

The *dump_load* parameter

Before setting dump_load to “on,” create function strings for the rs_dumpdb and rs_dumptran functions. Replication Server does not generate function strings for these functions in the system-provided classes or in derived classes that inherit from these classes.

The *save_interval* configuration parameter

Set save_interval to save transactions in the DSI queue that can be used to resynchronize a database after it has been restored from backups. Setting a save interval is also useful when you set up a warm standby of a database that holds replicate data or receives replicated functions. You can use sysadmin restore_dsi_saved_segments to restore backlogged transactions.

Network-based security parameters

- Both ends of a connection must use compatible Security Control Layer (SCL) drivers with the same security mechanisms and security features. The data server must support set proxy or an equivalent command.

  It is the replication system Administrator’s responsibility to choose and set security features for each server. Replication Server does not query the security features of remote servers before attempting to establish a connection. Connections fail if security features at both ends of the connection are not compatible.

- alter connection modifies network-based security settings for an outgoing connection from Replication Server to a target data server. It overrides default security parameters set with configure replication server.

- If unified_login is set to “required,” *only* the replication system Administrator with "sa" permission can log in to the Replication Server without a credential. If the security mechanism should fail, the replication system Administrator can log in to Replication Server with a password and disable unified_login.

- A Replication Server can have more than one security mechanism; each supported mechanism is listed in the *libtcl.cfg* file under SECURITY.
- Message encryption is a costly process with severe performance penalties. In most instances, it may be wise to set msg_confidentiality "required" only for certain connections. Alternatively, choose a less costly security feature, such as msg_integrity.

Using *alter connection* to change ERSSD maintenance passwords

- You can change ERSSD maintenance user passwords using the existing alter connection command:

```
alter connection to data_server.database
set password to password
```

- If your Replication Server is using ERSSD and the

```
data_server.database
```

  match the ERSSD names, using alter connection and set password updates the rs_maintusers table, issues sp_password at ERSSD, and updates the configuration file line RSSD_maint_pw.

Permissions

alter connection requires "sa" permission.

See also

admin show_connections, admin who, alter connection, configure replication server, create error class, create function string class, drop connection, resume connection, set proxy, suspend connection

alter database replication definition

Description

Changes an existing database replication definition.

Syntax

```
alter database replication definition db_repdef
          with primary at srv.db
          { [ not ] replicate DDL | [ not ] replicate setname setcont }
          [ with dsi_suspended ]
```

```
setname ::= { tables | functions | transactions | system procedures }
```

```
setcont ::= [ in ( [ owner1. ] name1 [, [ owner2. ] name2 [, ...] ])
```

Parameters

db_repdef
Name of the database replication definition.

server_name.db
Name of the primary server/database combination. For example: TOKYO.dbase.

[not] replicate DDL
Tells Replication Server whether or not to send DDL to subscribing databases. If "replicate DDL" is not included, or the clause includes "not," DDL is not sent to the replicate database.

[not] replicate *setname setcont*
Tells Replication Server whether or not to send objects in the *setname* category to the replicate database.

If *setcont* is omitted, Replication Server replicates all (or not replicate any if not is included) objects in the same *setname* category.

owner
An owner of a table or a user who executes a transaction. Replication Server does not process owner information for functions or system procedures.

You can replace *owner* with a space surrounded by single quotes or with an asterisk.

- A space (' ') – indicates no owner.
- An asterisk (*) – indicates all owners. Thus, for example, *.publisher means all tables named publisher, regardless of owner.

name

The name of a table, function, transaction, or system procedure.

You can replace *name* with a space surrounded by single quotes or with an asterisk.

- A space (‘ ‘) – indicates no name. For example, maintuser.’ ‘ means all unnamed maintenance user transactions.
- An asterisk (*) – indicates all names. Thus, for example, robert.* means all tables (or transactions) owned by robert.

with dsi_suspended

Tells the replicate Replication Server to suspend the replicate DSI. Can be used to signal need to resynchronize databases.

Examples

Changes the database replication definition rep_1C to filter out table2. The replicate DSI will be suspended:

```
alter database replication definition rep_1C
    with primary at PDS.pdb
    not replicate tables in (table2)
    with dsi_suspended
```

Usage

- When alter database replication definition is executed, Replication Server writes an rs_marker to the inbound queue. alter database replication definition does not take affect until the marker reaches the DIST, which gives the DIST time to incorporate the changes in the Database Subscription Resolution Engine (DSRE).
- Altering a database replication definition may desynchronize the primary and replication databases. See the *Replication Server Administration Guide Volume 1* for instructions for resynchronizing databases.

See also

create database replication definition, drop database replication definition

alter function

Description

Adds parameters to a user-defined function.

Syntax

alter function *table_rep_def.function_name*
add parameters *@param_name datatype*
[, *@param_name datatype*]...

Parameters

table_rep_def
The name of the replication definition upon which the user-defined function operates.

function_name
The name of the user-defined function to be altered.

@param_name
The name of a parameter to be added to the user-defined function's parameter list. The parameter name must conform to the rules for identifiers and must be preceded by an @ sign.

datatype
The datatype of the parameter. See "Datatypes" on page 21 for a list of the datatypes and their syntax. The parameter cannot be text, unitext, raw object, or image.

Examples

```
alter function publishers_rep.upd_publishers
 add parameters @state char(2)
```

Adds an integer parameter named state to the upd_publishers function for the publishers_rep replication definition.

Usage

- Before executing alter function, quiesce the replication system. You can use Replication Server Manager or the procedure described in the *Replication Server Troubleshooting Guide* to quiesce the system.
- A user-defined function can have up to 255 parameters.
- Altering functions during updates can cause unpredictable results. The affected data should be quiescent before you alter the function.
- After altering a user-defined function, you may also have to alter function strings that use the new parameters.
- When you alter a user-defined function for a replication definition, it is altered for all replication definitions of the primary table.
- Do not use alter function for replicated functions. Use alter function rep def instead. alter function is used only for the asynchronous stored procedures described in Chapter 6, "Adaptive Server Stored Procedures."

Permissions	alter function requires “create object” permission.
See also	admin quiesce_check, alter function string, create function, create function string, drop function, drop function string

alter function replication definition

Description

Changes an existing function replication definition.

Syntax

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

Parameters

function_rep_def
The name of the function replication definition to be altered.

deliver as
Specifies the name of the stored procedure to execute at the database where you are delivering the replicated function. *proc_name* is a character string of up to 200 characters. If you do not use this optional clause, the function is delivered as a stored procedure with the same name as the function replication definition.

add
Specifies additional parameters and their datatypes for the function replication definition.

@param_name
The name of a parameter to be added to the list of replicated parameters or searchable parameters. Each parameter name must begin with a @ character.

datatype
The datatype of the parameter you are adding to a parameter list. See "Datatypes" on page 21 for a list of supported datatypes and their syntax. Adaptive Server stored procedures and function replication definitions may not contain parameters with the text, unitext, and image datatypes.

add searchable parameters
Specifies additional parameters that can be used in the where clauses of the define subscription or define subscription command.

send standby
In a warm standby application, specifies whether to send all parameters in the function (send standby all parameters) or just those specified in the replication definition (send standby replication definition parameters) to a standby database. The default is send standby all parameters.

Examples

Example 1 Adds three parameters to the titles_frep function replication definition: a varchar parameter named @notes, a datetime parameter named @pubdate, and a bit parameter named @contract:

```
alter function replication definition titles_frep
 add @notes varchar(200), @pubdate datetime,
 @contract bit
```

Example 2 Adds the @type and @pubdate parameters to the list of searchable parameters in the titles_frep function replication definition:

```
alter function replication definition titles_frep
 add searchable parameters @type, @pubdate
```

Example 3 Changes the titles_frep function replication definition to be delivered as the newtitles stored procedure at the destination database, typically the primary database (used for request function delivery):

```
alter function replication definition titles_frep
 deliver as 'newtitles'
```

Usage

- alter function replication definition changes a function replication definition by adding replicated parameters, adding searchable parameters, specifying whether to send all parameters to the warm standby, or specifying a different name for the stored procedure to execute in the destination database.
- The name, parameters, and datatypes you specify for a function replication definition you are altering must match the stored procedure you are replicating. You can specify only those parameters you are interested in replicating.
- You must execute alter function replication definition at the Replication Server that manages the primary database (where you created the function replication definition).
- A parameter name must not appear more than once in any clause.
- If you are adding parameters, coordinate alter function replication definition with distributions for the function replication definition. Follow the steps in “Procedure to alter a function replication definition” on page 127 to avoid errors.
- You can use the optional deliver as clause to specify the name of the stored procedure to execute at the destination database where you are delivering the replicated function. Typically, you use this option in request function delivery. For more information, see create function replication definition.

  See the *Replication Server Administration Guide Volume 1* for more information on alter function replication definition.

Procedure to alter a function replication definition

❖ **Altering a function replication definition**

1 Quiesce the replication system using Sybase Central's Replication Manager plug-in or the procedure described in the *Replication Server Troubleshooting Guide*.

  Ideally, you should first quiesce primary updates and ensure that all primary updates have been processed by the replication system. If you are unable to do that, then old updates in the primary log will not have values for new parameters, and the replication system will use nulls instead. You may need to take this into account when altering function strings in step 4 below.

2 Alter the stored procedure at the primary and the replicate sites.

3 Alter the function replication definition. Wait for the modified function replication definition to arrive at the replicate sites.

4 If necessary, alter any function strings pertaining to the function replication definition. Wait for the modified function strings to arrive at the replicate sites.

5 If necessary, modify subscriptions on the function replication definition at replicate sites. To modify a subscription, drop it and re-create it using drop subscription and create subscription (with no materialization option).

  Altering a replication definition does not affect current subscriptions. If new parameters are added to the function replication definition, they are replicated with any new updates for all existing subscriptions.

6 Resume updates to the data at the primary database.

Permissions

alter function replication definition requires "create object" permission.

See also

alter function string, create function replication definition, drop function replication definition

alter function string

Description

Replaces an existing function string.

Syntax

```
alter function string [replication_definition.]function[;function_string]
          for function_class
          [scan 'input_template']
          [output
               {language 'lang_output_template' | rpc 'execute procedure
          [@param_name=]{constant |?variable!mod?}
               [, [@param_name=]{constant |?variable!mod?}]...' |
          writetext [use primary log | with log |
          no log] |
          none}]
```

Usage

- alter function string is the same as create function string, except that it executes drop function string first. The function string is dropped and re-created in a single transaction to prevent errors that would result from missing function strings.
- Alter function strings for functions with class scope at the primary site for the function string class. See create function string class for more information about the primary site for a function-string class.
- Alter function strings for functions with replication definition scope, including user-defined functions, at the site where the replication definition was created. Each replication definition has its own set of function strings.
- For rs_select, rs_select_with_lock, rs_datarow_for_writetext, rs_get_textptr, rs_textptr_init, and rs_writetext function strings, Replication Server uses the *function_string* name to determine which string to alter. If a *function_string* name was provided when the function string was created, you must specify it with alter function string so that the function string to be altered can be found.
- See create function string for more information about alter function string, including descriptions of keywords and options.
- To restore the default function string for a function, omit the output clause.

Permissions

alter function string requires "create object" permission.

See also

alter connection, create connection, create function, create function string, create function string class, define subscription, drop function string

alter function string class

Description

Alters a function-string class, specifying whether it should be a base class or a derived class.

Syntax

alter function string class *function_class*
    set parent to {*parent_class* | null}

Parameters

function_class
The name of an existing function-string class to be altered.

set parent to
Designates an existing class as a parent for the class you are altering; or, with the null keyword, designates that the class should be a base class.

parent_class
The name of an existing function-string class you designate as the parent class for a new derived class. rs_sqlserver_function_class may not be used as a parent class.

null
Specifies that the class should be a base class.

Examples

Example 1 Specifies that sqlserver2_function_class should become a derived class, inheriting function strings from the parent class rs_default_function_class:

```
alter function string class
 sqlserver2_function_class
 set parent to rs_default_function_class
```

Example 2 Specifies that the derived function-string class named rpc_xact should be a base class:

```
alter function string class rpc_xact
 set parent to null
```

Usage

- Use alter function string class to change a derived function-string class to a base class, to change the parent class of a derived class, or to change a base class to a derived class.
- The primary site for a derived class is the same as its parent class. Alter derived classes at the primary site of the parent class. However, if the parent class is a system-provided class, rs_default_function_class or rs_db2_function_class, the primary site for the derived class is the Replication Server where you created the derived class.

- See create function string class for more information about alter function string class.
- For more information about function-string classes, function strings, and functions, see the *Replication Server Administration Guide Volume 2*.
- Replication Server distributes the altered function-string class to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

Permissions

alter function string class requires "sa" permission.

See also

alter connection, create connection, create function, create function string, create function string class, drop function string class

alter logical connection

Description

Disables or enables the Distributor thread for a logical connection, changes attributes of a logical connection, and enables or disables replication of truncate table to the standby database.

Syntax

```
alter logical connection
        to logical_ds.logical_db {
        set distribution {on | off} |
        set logical_database_param to 'value' }
```

Parameters

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

distribution on
Enables the Distributor thread for the logical connection.

distribution off
Disables the Distributor thread for the logical connection.

logical_database_param
The name of a configuration parameter that affects logical connections. Table 3-15 describes the parameters you can set with alter logical connection.

value
A setting for a configuration parameter that matches the parameter. *value* is a character string.

Table 3-15: Configuration parameters affecting logical connections7

logical_database_param	value
dist_stop_unsupported_cmd	Use dist_stop_unsupported_cmd to set DIST to suspend itself or to continue running when it encounters commands not supported by downstream Replication Server. When dist_stop_unsupported_cmd is on, DIST suspends itself if a command is not supported by downstream Replication Server. If it is off, DIST ignores the unsupported command. Regardless of dist_stop_unsupported_cmd parameter's setting, Replication Server always logs an error message when it sees the first instance of a higher version command that cannot be sent over to a lower version Replication Server. Default: off
materialization_save_interval	Materialization queue save interval. This parameter is only used for standby databases in a warm standby application. Default: "strict" for standby databases

logical_database_param	**value**
replicate_minimal_columns	Specifies whether Replication Server should send all replication definition columns for all transactions or only those needed to perform update or delete operations at the standby database. Values are “on” and “off.” Replication Server uses this value in standby situations only when a replication definition does not contain a “send standby” option with any parameter. Otherwise, Replication Server uses the value of the “replicate minimal columns” or “replicate all columns” parameter in the replication definition. Default: on
save_interval	The number of minutes that the Replication Server saves messages after they have been successfully passed to the destination data server. See the *Replication Server Administration Guide Volume 2* for details. Default: 0 minutes
send_standby_repdef_cols	Specifies which columns Replication Server should send to the standby database for a logical connection. Overrides “send standby” options in the replication definition that tell Replication Server which table columns to send to the standby database. Values are: • on – send only the table columns that appear in the matching replication definition. Ignore the “send standby” option in the replication definition. • off – send all table columns to the standby. Ignore the “send standby” option in the replication definition. • check_repdef – send all table columns to the standby based on “send standby” option. Default: check_repdef
send_truncate_table	Specifies whether to enable or disable replication of truncate table to standby database. Values are: • on – enables replication of truncate table to standby database. This is the default for warm standby applications created in Adaptive Server version 11.5 or later. • off – disables replication of truncate table to standby database. This is the default for warm standby applications where the active and/or standby databases were created in pre-11.5 SQL Server and have been upgraded to Adaptive Server version 11.5 or later.

Examples

Example 1 Disables the distributor thread for the LDS.pubs2 logical connection:

```
alter logical connection to LDS.pubs2
 set distribution off
```

Example 2 Changes the save interval for the LDS.pubs2 logical connection to "0," allowing messages in the DSI queue for the logical connection to be deleted:

```
alter logical connection to LDS.pubs2
 set save_interval to '0'
```

Example 3 Enables the replication of truncate table to the standby database:

```
alter logical connection to LDS.pubs2
 set send_truncate_table to 'on'
```

Usage

- To copy truncate table to a warm standby database, set the send_truncate_table option to "on."
- Set the send_truncate_table option to "on" only when both the active and warm standby databases are at Adaptive Server version 11.5 or later.
- If you specify the send_truncate_table to on clause, Replication Server copies the execution of truncate table to the warm standby database for all tables marked for replication.
- If the warm standby database was created before you installed or upgraded to Replication Server version 11.5 or 12.0, send_truncate_table is set to "off" by default. Make "off" the default for send_truncate_table to preserve compatibility with existing warm standby applications. You must set send_truncate_table to "on" for Replication Server to copy truncate_table to the warm standby database.
- Use the alter logical connection command to disable the Distributor thread after you set up a warm standby application. When you add a database to the replication system, Replication Server creates a Distributor thread to process subscriptions for the data.
- Use the set distribution off clause to disable the Distributor thread for a logical connection. Use this option when you have set up a warm standby for a database but there are no subscriptions for the data in the database, and if the database is not a source of replicated stored procedure execution. Such a logical database may be a warm standby application that does not involve normal replication, or it may be a logical replicate database.
- Use set distribution on to start the Distributor thread for a logical connection after you disable it with set distribution off. Do this to create replication definitions and subscriptions for the data in the logical database, or to initiate replicated stored procedures in the logical database.

- You can suspend or resume a Distributor thread for a physical or logical database connection using the suspend distributor and resume distributor commands.
- See the *Replication Server Administration Guide Volume 1 and Volume 2* for more information about setting up and managing warm standby applications.
- You can set parameters that affect all logical connections originating at the current Replication Server with the configure replication server command.
- The save_interval parameter for a logical connection is set to 'strict,' by default, when the logical connection is created. This ensures that messages are not deleted from DSI queues before they are applied to the standby database.

  If the standby database is not available for a long period of time, Replication Server's queues may fill. To avoid this, change save_interval from 'strict' to "0" (minutes). This allows Replication Server to delete the queues.

  Warning! The save_interval parameter affects only the DSI queue. The materialization_save_interval parameter affects only currently existing materialization queues. They should *only* be reset under serious conditions caused by a lack of stable queue space. Resetting it (from 'strict' to a given number of minutes) may lead to message loss at the standby database. Replication Server cannot detect this type of loss; you must verify the integrity of the standby database yourself.

- The materialization_save_interval parameter for a logical connection is set to 'strict,' by default, when the logical connection is created. This ensures that messages are not deleted from materialization queues before they are applied to the standby database.

  If the standby database is not available for a long period of time, Replication Server's queues may fill. To avoid this, change materialization_save_interval from 'strict' to "0" (minutes). This allows Replication Server to delete the queues.

See also

admin logical_status, configure replication server, create logical connection, resume distributor, suspend distributor

alter partition

Description

Changes the size of a partition.

Syntax

alter partition *logical_name* [expand [size = *size*]]

Parameters

logical_name

A name for the partition. The name must conform to the rules for identifiers. The name is also used in the drop partition and create partition commands.

expand

Specifies that the partition is to increase in size.

size

Specifies the number of megabytes the partition is to increase. The default value is 2MB.

Examples

Example 1 This example increases the size of logical partition P1 by 50MB:

```
alter partition P1 expand size = 50
```

Example 2 This example increases the size of logical partition P2 by 2MB:

```
alter partition P2
```

Usage

- alter partition allows users to expand a currently used partition to a larger size. This function is useful when Replication Server needs more disk space and there is still space available in the same disk of the existing partition.
- In case of insufficient physical disk space, alter partition aborts and an error message displays. The allocated space for the partition is the same as before the command was applied.
- The maximum size that can be allocated to a partition is 1TB, which is approximately 1,000,000MB.

Permissions

Only the "sa" user can execute alter partition.

See also

admin disk_space, create partition, drop partition

alter queue

Description

Specifies the behavior of the stable queue that encounters a large message of greater than 16K bytes. Applicable only when the Replication Server version is 12.5 or later and the site version is 12.1 or earlier.

Syntax

alter queue, *q_number*, *q_type*, set sqm_xact_with_large_msg [to]
{skip | shutdown}

Parameters

q_number
The queue number of the stable queue.

q_type
The queue type of the stable queue. Values are “0” for outbound queues and “1” for inbound queues.

sqm_xact_with_large_msg {skip | shutdown}
Specifies that the SQM should skip or shut down when encountering a message larger than 16K bytes.

Examples

Specifies that outbound queue #2 shuts down if it is passed a large message:

```
alter queue, 2, 0, set sqm_xact_with_large_msg to
    shutdown
```

Usage

alter queue fails if the site version is 12.5 or later.

Permissions

alter queue requires "sa" permission.

See also

alter route, resume queue, resume route

alter replication definition

Description

Changes an existing replication definition.

Syntax

```
alter replication definition replication_definition
{with replicate table named table_owner.]'table_name' |
add colum_name [as replicate_column_name]
                    [datatype [null | not null]]
                    [map to published_datatype],... |
alter columns with column_name
                    [as replicate_column_name],...|
alter columns with column_name
                    datatype [null | not null]
                    [map to published_datatype],...|
add primary key column_name [, column_name]... |
drop primary key column_name [, column_name]... |
add searchable columns column_name [, column_name]... |
drop searchable columns column_name [, column_name]... |
send standby [off | {all | replication definition} columns] |
replicate {minimal | all} columns |
replicate_if_changed column_name [, column_name]... |
always_replicate column_name [, column_name]...}
```

Parameters

replication_definition
The name of the replication definition to alter.

with replicate table named
Specifies the name of the table at the replicate database. *table_name* is a character string of up to 200 characters. *table_owner* is an optional qualifier for the table name, representing the table owner. Data server operations may fail if actual table owners do not correspond to what you specify in the replication definition.

add columns *column_name*
Specifies additional columns and their datatypes for the replication definition. *column_name* is the name of a column to be added to the replicated columns list. The column name must be unique for a replication definition.

Also add columns *declared_column_name*. See “Using column-level datatype translations” on page 142.

as *replicate_column_name*
For columns you are adding to the replication definition, specifies a column name in a replicate table into which data from the primary column will be replicated. *replicate_column_name* is the name of a column in a replicate table that corresponds to the specified column in the primary table. Use this clause when the replicate and primary columns have different names.

datatype

The datatype of the column you are adding to a replication definition column list or the datatype of an existing column you are altering. See "Datatypes" on page 21 for a list of supported datatypes and their syntax.

If a column is listed in an existing replication definition for a primary table, subsequent replication definitions for the same primary table must specify the same datatype.

Use as *declared_datatype* if you are specifying a column-level datatype translation for the column. A declared datatype must be a native Replication Server datatype or a datatype definition for the primary datatype.

null or not null

Applies only to text, unitext, image, and rawobject columns. Specifies whether a null value is allowed in the replicate table. The default is not null, meaning that the replicate table does not accept null values.

The null status for each text, unitext, image, and rawobject column must match for all replication definitions for the same primary table, and must match the settings in the actual tables. Specifying the null status is optional if an existing replication definition of the same primary table has text, unitext, image, or rawobject columns.

alter columns *column_name*

Specifies columns and their datatypes to alter in the replication definition. *column_name* is the name of a column to be changed. The column name must be unique for a replication definition.

Use alter columns *declared_column_name* when specifying a column-level datatype translation.

map to *published_datatype*

Specifies the datatype of a column after a column-level datatype translation. *published_datatype* must be a Replication Server native datatype or a datatype definition for the published datatype.

add/drop primary key

Used to add or remove columns from the primary keys column list.

Replication Server depends on primary keys to find the correct rows at the replicate or standby table. To drop all primary key columns, first alter the corresponding replication definition to add the new primary keys, then drop the old primary key columns in the table. If all primary keys are missing, the DSI will shut down. See create replication definition for additional information on primary keys.

add searchable columns *column_name*
: Specifies additional columns that can be used in where clauses of the create subscription or define subscription command. *column_name* is the name of a column to add to the searchable columns list. The same column name must not appear more than once in each clause.

  You cannot specify text, unitext, image, rawobject, rawobject in row or encrypted columns as searchable columns.

drop searchable columns *column_name*
: Specifies columns to remove from the searchable column list. You can remove columns from the searchable column list only if they are not used in subscription or article where clauses.

send standby
: Specifies how to use the replication definition in replicating into a standby database in a warm standby application. See "Replicating into a standby database" on page 143 for details on using this clause and its options.

replicate minimal columns
: Sends (to replicate Replication Servers) only those columns needed to perform update or delete operations at replicate databases. To replicate all columns, use replicate all columns.

replicate_if_changed
: Specifies text, unitext, image, or rawobject columns to be added to the replicate_if_changed column list. When multiple replication definitions exist for the same primary table, using this clause to change one replication definition changes all replication definitions of the same primary table.

always_replicate
: Specifies text, image, or rawobject columns to be added to the always_replicate column list. When multiple replication definitions exist for the same primary table, using this clause to change one replication definition changes all replication definitions of the same primary table.

Examples

Example 1 Adds state as a searchable column to the authors_rep replication definition:

```
alter replication definition authors_rep
 add searchable columns state
```

Example 2 Changes the titles_rep replication definition to specify that only the minimum number of columns will be sent for delete and update operations:

```
alter replication definition titles_rep
 replicate minimal columns
```

Example 3 Changes the titles_rep replication definition to specify that the replication definition can be subscribed to by a replicate table called copy_titles owned by the user “joe”:

```
alter replication definition titles_rep
 with replicate table named joe.'copy_titles'
```

Example 4 Changes the pubs_rep replication definition to specify that the primary column pub_name will replicate into the replicate column pub_name_set:

```
alter replication definition pubs_rep
alter columns with pub_name as pub_name_set
```

Example 5 Introduces a column-level translation that causes hire_date column values to be translated from rs_db2_date (primary) format to the native datatype smalldatetime (replicate) format:

```
alter replication definition employee_repdef
alter columns with hire_date as rs_db2_date
map to smalldatetime
```

Usage

- Use the alter replication definition command to change a replication definition by:
  - Adding or dropping primary keys
  - Changing the name of a target replicate table
  - Changing the names of target replicate columns
  - Adding columns and indicating the names of corresponding target replicate columns
  - Adding or dropping searchable columns
  - Changing replication definition usage by warm standby applications
  - Changing column datatypes
  - Changing between replicating all or minimal columns
  - Changing replication status for text, unitext, image, or rawobject columns
  - Introducing or removing a column-level datatype translation.

- Execute alter replication definition at the primary site for the replication definition.
- For a database replication definition to replicate encrypted columns without using a table level replication definition, you must define the encryption key for the encrypted columns with INIT_VECTOR NULL and PAD NULL
- If you use more than one version of Replication Server (for example, Replication Server version 12.0 and version 11.0.x) and create multiple replication definitions for the same primary table, the first replication definition created, which has the same primary and replicate table names and does not include table owner name, is marked and propagated to pre-11.5 Replication Servers.

  If you alter a replication definition propagated to a pre-11.5 Replication Server so that it is no longer version 11.0.x compatible, and subscriptions exist to that replication definition from 11.0.x and earlier sites, you cannot use alter replication definition. If there are no subscriptions from pre-11.5 to the replication definition, the definition is dropped from pre-11.5 sites, and the oldest replication definition created for that table that is compatible with version 11.0.x is distributed to the Replication Server of an earlier version so subscriptions can be created against it. See create replication definition for more information about working with replication definitions in a mixed-version environment.
- alter replication definition affects the version compatibility of a pre-12.0 replication definition if you change the datatype of a column to rawobject or rawobject in row or add a column with a datatype of rawobject or rawobject in row.

  Thus, for example, if you introduce a rawobject datatype into a replication definition compatible with a pre-12.0 version of Replication Server, the version of the replication definition will change and will no longer be compatible with the pre-12.0 Replication Server.
- See create replication definition for more information about the options in the alter replication definition command.

Adding columns

- If you add columns, coordinate alter replication definition with distributions for the replication definition. To avoid errors, follow the steps in “Procedure to alter a replication definition” on page 144.

- If a column you are adding to a replication definition contains an IDENTITY column, the maintenance user must be the owner of the table (or must be "dbo" or aliased to "dbo") at the replicate database in order to use the Transact-SQL identity_insert option. A primary table can contain only one IDENTITY column.

Altering column datatypes

- You cannot change the column datatype if it is used in a subscription or article where clause.
- You cannot change the rs_address datatype.
- You can change the column datatype to a text, untext, image, rawobject, or rawobject in row datatype only if it is not a primary key or searchable column.
- To change the published datatype of a column, you must specify both the declared datatype and the map to option.
- Projections require that datatype and nullability be consistent across all projections for a table for declared datatypes. This is not necessary for published datatypes.

Note Changes between a rawobject or rawobject in row and its base datatype for which only the current replication definition is affected do not affect all projections.

- See the *Replication Server Administration Guide Volume 1*, which describes how to change datatypes.
- Use column nullability changes only for text, unitext, image, and rawobject columns.

Using column-level datatype translations

- To effect column-level datatype translations, you must first set up and install the heterogeneous datatype support (HDS) objects as described in the *Replication Server Configuration Guide* for your platform.
- You cannot use text, unitext, image, or rawobject datatypes as a base datatype or a datatype definition or as a source or target of either a column-level or class-level translation.
- *declared_datatype* depends on the datatype of the value delivered to Replication Server:
  - If the Replication Agent delivers a base Replication Server datatype, *declared_datatype* is the base Replication Server datatype.

    - If the Replication Agent delivers any other datatype, *declared_datatype* must be the datatype definition for the original datatype in the primary database.

- *published_datatype* is the datatype of the value after a column-level translation, but before any class-level translation. *published_datatype* must be a Replication Server native datatype or a datatype definition for the datatype in another database.
- Columns declared in multiple replication definitions must use the same *declared_datatype* in each replication definition. *published_datatype* can differ.

Replicating all or minimal columns

- When you use replicate minimal option for a replication definition, data is sent to replicate Replication Servers for the minimum number of columns needed for delete or update operations. Specify replicate all columns to replicate all columns. See create replication definition for additional information about this feature.

Replicating into a standby database

- Replication Server does not require replication definitions to maintain a standby database in a warm standby application. Using replication definitions may improve performance in replicating into the standby database. You can create a replication definition just for this purpose for each table in the logical database.
- Use send standby with any option other than off to use this replication definition to replicate transactions for this table to the standby database. The replication definition's primary key columns and replicate minimal columns setting are used to replicate into the standby database. The options for this method include:
    - Use send standby or send standby all columns to replicate all primary table columns into the standby database.
    - Use send standby replication definition columns to replicate only the replication definition's columns into the standby database.

- Use send standby off to indicate that no single replication definition for this table should be used in replicating into the standby database. All the columns in the table are replicated into the standby database, and the union of all primary key columns in all replication definitions for the table is used in replicating into the standby database. The replicate_minimal_columns setting of the logical connection determines whether to send minimal columns or all columns for update and delete. See alter logical connection.

  If no replication definition exists for a table, all the columns in the table are replicated into the standby database and Replication Server constructs the primary key. In this case, replicate_minimal_columns is on.

Procedure to alter a replication definition

❖ **Altering a replication definition**

1. Quiesce the replication system. You can use Replication Server Manager or the procedure described in the *Replication Server Administration Guide Volume 1* to quiesce the system.

   Ideally, you should first quiesce primary updates and ensure that all primary updates have been processed by the replication system. If you are unable to do that, then old updates in the primary log will not have values for new columns, and the replication system will use nulls instead. You may need to take this into account when altering function strings in step 4 below.

2. Shut down the RepAgent for the primary database.
3. Alter the table at the primary site and the replicate sites, if this is why you are altering the replication definition.
4. Alter the replication definition as many times as needed. Wait for the modified replication definition to arrive at the replicate sites.
5. If necessary, alter any function strings pertaining to the replication definition. Wait for the modified function strings to arrive at the replicate sites.
6. If necessary, modify subscriptions on the replication definition at destination sites. To modify a subscription, drop it and re-create it using the drop subscription and create subscription commands.

   Altering a replication definition does not affect current subscriptions. If new columns are added to the replication definition, they are replicated with any new updates for all existing subscriptions.

7. If the dynamic SQL feature is enabled for the connection to the replicate database, suspend and resume the connection to clear out the old prepared statements.
8. Start up the RepAgent for the primary database.
9. Resume updates to the primary table.

Altering replication status for *text, unitext, image*, and *rawobject* columns

Certain restrictions apply to replication status for text, unitext, image, and rawobject columns in replicated and warm standby databases. See "Replicating text, unitext, image, or rawobject datatypes" on page 236.

❖ **Changing the replication status of a *text*, *unitext*, *image*, or *rawobject* column from *replicate_if_changed* to *always_replicate***

1. Stop updates to the primary table.
2. Wait for transactions that modify the primary table and have text, unitext, image, and rawobject columns with a replicate_if_changed status to arrive at the replicate sites.
3. Use sp_setrepcol to change the status of the column at the Adaptive Server to always_replicate.
4. Use alter replication definition to change the status of the column to always_replicate. Wait for the modified replication definition to arrive at the replicate sites.
5. Resume updates to the primary table.

❖ **Changing the replication status of a *text*, *untext*, *image*, or *rawobject* column from *always_replicate* to *replicate_if_changed***

1. Use alter replication definition to change the status of the column to replicate_if_changed. Wait for the modified replication definition to arrive at the replicate sites.
2. Use sp_setrepcol to change the status of the column at the Adaptive Server to replicate_if_changed.

Note When changing the status from always_replicate to replicate_if_changed, there is no need to stop updates to the primary table because the change in status does not result in a RepAgent error.

Permissions

alter replication definition requires "create object" permission.

See also

alter function string, create replication definition, drop replication definition, set autocorrection

alter route

Description

Changes the attributes of a route from the current Replication Server to a remote Replication Server.

Syntax

```
alter route to dest_replication_server {
          set next site [to] thru_replication_server |
          set username [to] 'user' set password [to] 'passwd' |
          set password [to] 'passwd' |
          set route_param [to] 'value' |
          set security_param [to] 'value' |
          set security_services [to] 'default'}
```

Parameters

dest_replication_server
The name of the destination Replication Server whose route you are altering.

thru_replication_server
The name of an intermediate Replication Server through which messages for the destination Replication Server will be passed.

user
The login name to use for the route.

passwd
The password to use with the login name.

route_param
A parameter that affects routes. Refer to Table 3-16 for a list of parameters and values.

value
A setting for *route_param*. It is a character string.

Table 3-16: Configuration parameters affecting routes

route_param	value
disk_affinity	Specifies an allocation hint for assigning the next partition. Enter the logical name of the partition to which the next segment should be allocated when the current partition is full. Default: off
rsi_batch_size	The number of bytes sent to another Replication Server before a truncation point is requested. Default: 256KB Minimum: 1KB Maximum: 128MB
rsi_fadeout_time	The number of seconds of idle time before Replication Server closes a connection with a destination Replication Server. Default: -1 (specifies that Replication Server will not close the connection)

route_param	value
rsi_packet_size	Packet size, in bytes, for communications with other Replication Servers. The range is 1024 to 16384 bytes. Default: 2048 bytes
rsi_sync_interval	The number of seconds between RSI synchronization inquiry messages. The Replication Server uses these messages to synchronize the RSI outbound queue with destination Replication Servers. Values must be greater than 0. Default: 60 seconds
rsi_xact_with_large_msg	Specifies route behavior if a large message is encountered. This parameter is applicable only to direct routes where the site version at the replicate site is 12.1 or earlier. Values are “skip” and “shutdown.” Default: shutdown
save_interval	The number of minutes that the Replication Server saves messages after they have been successfully passed to the destination Replication Server. See the *Replication Server Administration Guide Volume 2* for details. Default: 0 minutes

security_param
specifies the name of a security parameter. For a list and description of security parameters that can be set with alter route, refer to Table 3-27 on page 240.

set security_services [to] 'default'
resets all network-based security features for the connection to match the global settings of your Replication Server.

Examples

Example 1 In examples 1 and 2, direct routes exist from the Tokyo Replication Server (TOKYO_RS) to the San Francisco Replication Server (SF_RS) and to the Sydney Replication Server (SYDNEY_RS). The following commands change one direct route into an indirect route, so that TOKYO_RS passes messages destined for SYDNEY_RS through SF_RS.

Entered at SF_RS, this command creates a direct route to SYDNEY_RS that will be used by the new indirect route:

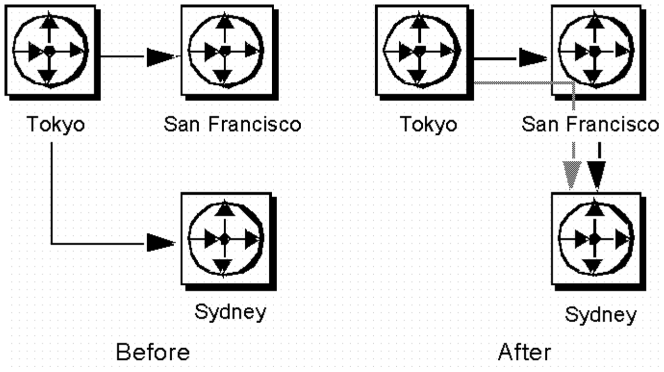
```
create route to SYDNEY_RS
    set username SYDNEY_rsi_user
    set password SYDNEY_rsi_passwd
```

Example 2 Entered at TOKYO_RS, this command changes the direct route from TOKYO_RS to SYDNEY_RS to an indirect route, specifying SF_RS as an intermediate Replication Server:

```
alter route to SYDNEY_RS
 set next site SF_RS
```

Figure 3-1 shows the routes before and after changing the routing scheme.

Figure 3-1: Before and after altering routing in examples 1 and 2



Examples 3 and 4 change the routing so that TOKYO_RS sends messages directly to SYDNEY_RS again, instead of passing them through SF_RS.

Example 3 Entered at TOKYO_RS, this command changes the route from TOKYO_RS to SYDNEY_RS from an indirect route to a direct route:

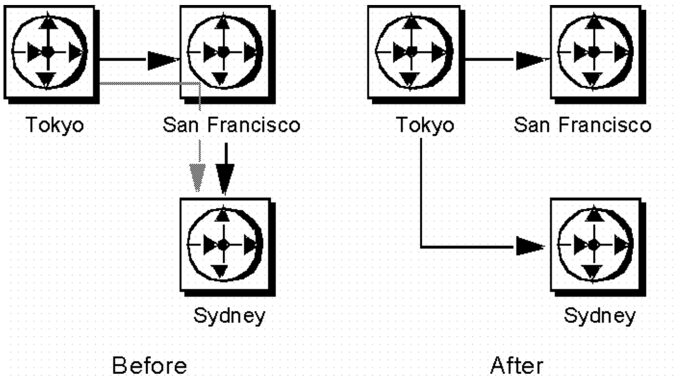
```
alter route to SYDNEY_RS
 set username SYDNEY_rsi
 set password SYDNEY_rsi_passwd
```

Example 4 Entered at SF_RS, this command removes the direct route from SF_RS to SYDNEY_RS:

```
drop route to SYDNEY_RS
```

Together, the commands in examples 3 and 4 cancel the effects of examples 1 and 2. Figure 3-2 shows the routes after the second set of commands is entered.

Figure 3-2: After altering routing



In example 5, direct routes exist from TOKYO_RS to SYDNEY_RS and from SYDNEY_RS to SF_RS, and an indirect route exists from TOKYO_RS to SF_RS, through SYDNEY_RS. This example changes this routing scheme so that TOKYO_RS passes messages destined for SF_RS through a different Replication Server, LA_RS in Los Angeles.

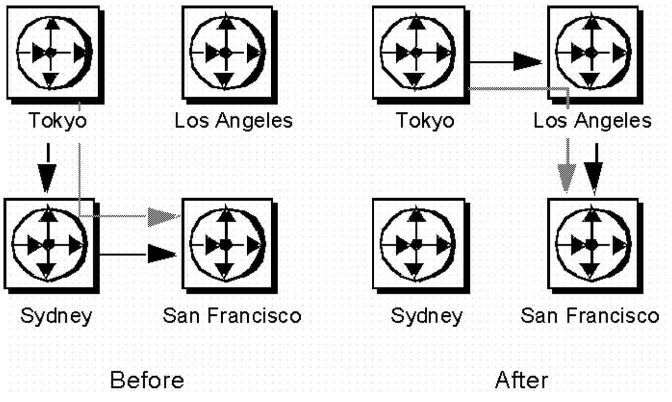
Example 5 Entered at TOKYO_RS, this command changes the intermediate Replication Server for the indirect route to LA_RS instead of SYDNEY_RS.

```
alter route to SF_RS
 set next site LA_RS
```

Before the route can be altered, direct routes must have been created from TOKYO_RS to LA_RS and from LA_RS to SF_RS.

Figure 3-3 shows the routes before and after the necessary commands have been entered. (Direct routes to and from SYDNEY_DS are not shown because you may have dropped them.)

Figure 3-3: Before and after necessary commands



Example 6 Entered at TOKYO_RS, this command changes the password for the direct route from TOKYO_RS to LA_RS. The new password is “LApass.”

```
alter route to LA_RS
 set password LApass
```

Before you change the password for the direct route, you must suspend the route using suspend route.

Example 7 Sets the security service to DCE for the route to LA_RS:

```
suspend route to LA_RS

 alter route to LA_RS
 set security_mechanism to 'dce'

 resume route to LA_RS
```

Usage

- Use alter route to change:
  - A direct route to an indirect route.
  - An indirect route to a direct route.
  - The next intermediate site in an existing route.
  - The password for the RSI user for an existing direct route.
  - A route configuration parameter.
  - A network-based security parameter.

    For an overview of routes, see the *Replication Server Administration Guide Volume 1*.

- Execute alter route at the Replication Server that is the source for a direct route.
- Use set next site *thru_replication_server* when you are changing a direct route into an indirect route, or when you are changing the intermediate site in an indirect route.
- If you are changing a direct route to an indirect route, you must first create direct routes from the source site to the intermediate site, and from the intermediate site to the destination site. Do this with create route.
- If you are changing the intermediate site in an indirect route, you must first create direct routes from the new intermediate site to the destination site, and from the new intermediate site to the destination site. Do this with create route.
- An indirect route may have one or more intermediate Replication Servers. For example, an indirect route from A_RS to D_RS may pass through intermediate sites B_RS and C_RS.
- To change an indirect route to a direct route, use alter route without the set next site clause, specifying the login name and password to use at the destination Replication Server. For example, an indirect route from A_RS->B_RS->C_RS changes to a direct route A_RS->C_RS.
- To exchange one intermediate site for the next intermediate site, execute alter route with the set next site clause. For example, an indirect route A_RS->B_RS->C_RS->D_RS changes to A_RS->C_RS->D_RS.
- You can set route parameters using the configure route or alter route parameter.
- Use suspend route to suspend activity on the route before altering it.

set password and *set username*

- Use set username *user* and set password *passwd* only when you are changing an indirect route to a direct route. You cannot change the user name or password for indirect routes; attempting to do so changes the indirect route to a direct route.
- Use set password *passwd* only when you are changing the password for a direct route. Before you change the password for a direct route, use suspend route.

Route parameters

- Setting a save interval allows the system to tolerate partition or stable queue failures at the destination Replication Server. Backlogged messages are sent to the destination Replication Server during recovery with the rebuild queues command.

  See the *Replication Server Administration Guide Volume 2* for detailed information about the save interval and stable queue recovery.
- Sybase recommends that you leave the rsi_batch_size, rsi_fadeout_time, rsi_packet_size, and rsi_sync_interval parameters at their default values to optimize performance.
- You must suspend the connection before altering a route parameter with alter route. After executing the alter route command, you must resume the route for the change to take effect.

Network-based security parameters

- Both ends of a route must use compatible Security Control Layer (SCL) drivers with the same security mechanisms and security features. It is the replication system Administrator's responsibility to choose and set security features for each server. The Replication Server does not query the security features of remote servers before attempting to establish a connection. Connections will fail if security features at both ends of the route are not compatible.
- alter route alters network-based security settings for an outgoing connection from Replication Server to a target Replication Server. Security parameters set by alter route override default values set by configure replication server.
- If unified_login is set to "required," *only* the "sa" user can log in to the Replication Server without a credential. If the security mechanism should fail, the "sa" user can then log in to Replication Server with a password and disable unified_login.
- A Replication Server can have more than one security mechanism; each supported mechanism is listed in the *libtcl.cfg* file under SECURITY.
- Message encryption is a costly process with severe performance penalties. In most instances, it is wise to set msg_confidentiality "on" only for certain connections. Alternatively, choose a less costly security feature, such as msg_integrity.

- You must suspend the connection before altering a security parameter with alter route. After you execute alter route, resume the route for the change to take effect.

Procedure to alter a route

Note If you are changing a configuration parameter, you only need to suspend the route before executing alter route.

---

1. Quiesce the replication system. For more detailed information, refer to the *Replication Server Troubleshooting Guide*.
2. Suspend log transfer with suspend log transfer at each Replication Server that manages a database with a RepAgent.
3. Execute the alter route command at the source Replication Server. You may alter as many routes as necessary.
4. Resume RepAgent connections to each RSSD and user database using resume log transfer.

   See the *Replication Server Administration Guide Volume 1* for complete procedures for altering routes.

Permissions

alter route requires "sa" permission.

See also

admin quiesce_check, admin quiesce_force_rsi, alter connection, alter logical connection, alter queue, configure connection, create logical connection, create replication definition, configure replication server, drop logical connection, create connection, create route, drop connection, drop route, resume log transfer, set proxy, suspend log transfer, suspend route

alter user

Description

Changes a user's password.

Syntax

```
alter user user
        set password {new_passwd | null}
        [verify password old_passwd]
```

Parameters

user

The login name whose password you want to modify.

new_passwd

The new password. It can be up to 30 characters long and include letters, numerals, and symbols. Case is significant. If the password contains spaces, enclose the password in quotation marks. When you create or alter a user login name, you must specify a password or "null." A null password lets a user log in immediately without being prompted for a password string.

verify password *old_passwd*

Enter your current password. Users who do not have "sa" permission must enter this clause.

Examples

The user with login name "louise" has changed her own password from "EnnuI" to "somNIfic":

```
alter user louise
 set password somNIfic
 verify password EnnuI
```

Usage

- If your Replication Server uses ERSSD, you can change the ERSSD primary user password using the alter user command:

  ```
  alter user user
  set password new_passwd
  ```

  If this user name matches the ERSSD primary user name, ERSSD updates the rs_users table, issues sp_password at ERSSD to change the password, and updates the configuration file line RSSD_primary_pw.

- Users with "sa" permission can omit the verify password clause. Other users must provide this clause in order to change their own passwords.

Permissions

alter user requires "sa" permission when altering another user's password.

See also

create user, drop user

assign action

Description

Assigns Replication Server error-handling actions to data server errors received by the DSI thread.

Syntax

```
assign action
        {ignore | warn | retry_log | log |
            retry_stop | stop_replication}
for error_class
        to data_server_error [, data_server_error]...
```

Parameters

ignore

Instructs Replication Server to ignore the error and continue processing. ignore should be used when the data server error code indicates a successful execution or an inconsequential warning.

warn

Instructs Replication Server to display a warning message in its log file without rolling back the transaction or interrupting execution.

retry_log

Instructs Replication Server to roll back the transaction and retry it. The number of retry attempts is set with alter connection. If the error continues after retrying, Replication Server writes the transaction in the exceptions log and executes the next transaction.

log

Instructs Replication Server to roll back the current transaction, log it in the exceptions log, and then execute the next transaction.

retry_stop

Instructs Replication Server to roll back the transaction and retry it. The number of retry attempts is set with the alter connection. If the error continues after retrying, Replication Server suspends replication for the database.

stop_replication

Instructs Replication Server to roll back the current transaction and suspend replication for the database. This action is equivalent to using suspend connection.

error_class

The error class name for which the action is being assigned.

data_server_error

A data server error number.

Examples

Instructs Replication Server to ignore data server errors 5701 and 5703:

```
assign action ignore
 for pubs2_db_err_class
 to 5701, 5703
```

Usage

- Use assign action to tell Replication Server how to handle errors returned by data servers. This command overrides any action previously assigned to a data server error.
- Execute assign action at the primary site where the create error class was executed.
- Assign actions for an error class before you create any distributions that use the error class. Assigning actions for an active distribution can lead to unpredictable results.
- If an error has no action assigned, the default action stop_replication is taken.
- Be sure to assign error actions that are appropriate for the error condition. For example, if you assign the ignore action to an error returned by the data server when a begin transaction command fails, the subsequent commit or rollback command may generate an unexpected error.
- Data servers return errors to Replication Server through the Client/Server Interfaces error-handling mechanism. Warnings and error messages are written to the Replication Server log file.
- Replication Server distributes error actions to qualifying sites through the replication system. The changes do not appear immediately because of normal replication system lag time.

Error actions with multiple errors

- When an operation results in multiple errors, Replication Server chooses the most severe action to perform for the set of errors. For example, if one error indicates that a transaction has been rolled back and is assigned the retry_log action, and another error indicates that the transaction log is full and is assigned the stop_replication action, a transaction that returns both errors causes Replication Server to perform the stop_replication action. The severity of the error actions, from least severe to most severe, are as follows:

  1. ignore
  2. warn
  3. retry_log
  4. log

5. retry_stop
6. stop_replication

Error actions for *rs_sqlserver_error_class*

- Predefined error actions for Adaptive Servers are provided with the rs_sqlserver_error_class error class.
- To assign different error actions in the rs_sqlserver_error_class, you must first choose a primary site for the error class. Log into the Replication Server at that site and create the error class using create error class.

Displaying error actions

- The rs_helperror stored procedure displays the Replication Server error actions mapped to a given data server error number.

Permissions

assign action requires "sa" permission.

See also

configure connection, create connection, create error class, drop error class, rs_helperror, suspend connection

check publication

Description

Finds the status of a publication and the number of articles the publication contains.

Syntax

check publication *pub_name*
    with primary at *data_server*.*database*

Parameters

pub_name
The name of the publication to check.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

Examples

Checks the status of the publication pubs2_pub, where the primary database is TOKYO_DS.pubs2:

```
check publication pubs2_pub
 with primary at TOKYO_DS.pubs2
```

Usage

- Use check publication to find the status of a publication and the number of articles the publication contains.

  See the *Replication Server Administration Guide Volume 1* for more information about publications.

- Execute check publication at the Replication Server that manages the replicate database or at the Replication Server that manages the primary database.
- If you execute check publication at the replicate Replication Server, the publication is checked at the primary Replication Server using the current user name and password. You must have the same login name and password at the primary Replication Server to display current information about the publication.
- To check subscription status, use check subscription. See check subscription for more information.

Messages returned by *check publication*

- When you execute check publication at a primary or replicate Replication Server, it returns one of these messages:

  ```
  Publication pub_name for primary database
  data_server.database is valid. The number of
  articles in the publication is number_articles.
  ```

```
Publication pub_name for primary database
data_server.database is invalid. The number of
articles in the publication is number_articles.
```

- When you execute check publication at a replicate Replication Server, it returns this message if it cannot contact the primary Replication Server:

```
Failed to get publication information from primary.
```

Permissions

Any user may execute this command. A user who enters this command at a replicate Replication Server must have the same login name and password in the primary Replication Server.

See also

check subscription, create publication, validate publication

check subscription

Description

Finds the materialization status of a subscription to a replication definition or a publication.

Syntax

check subscription *sub_name*
  for {*table_rep_def* | *function_rep_def* |
  [publication *pub_name* | database replication definition *db_repdef*]
  with primary at *data_server*.*database* }
  with replicate at *data_server*.*database*

Parameters

sub_name
The name of the subscription to check.

for *table_rep_def*
Specifies the name of the table replication definition the subscription is for.

for *function_rep_def*
Specifies the name of the function replication definition the subscription is for.

for publication *pub_name*
Specifies the name of the publication the subscription is for.

database replication definition *db_repdef*
Specifies the name of the database replication definition the subscription is for.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database. Include this clause only for a subscription for a publication.

with replicate at *data_server.database*
Specifies the location of the replicate data. If the replicate database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

Examples

Example 1 Checks the status of the subscription titles_sub for the replication definition titles_rep, where the replicate database is SYDNEY_DS.pubs2:

```
check subscription titles_sub
 for titles_rep
 with replicate at SYDNEY_DS.pubs2
```

Example 2 Checks the status of the subscription pubs2_sub for the publication pubs2_pub, where the primary database is TOKYO_DS.pubs2 and the replicate database is SYDNEY_DS.pubs2:

```
check subscription pubs2_sub
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replicate at SYDNEY_DS.pubs2
```

Usage

- Use check subscription to find the status of a subscription during subscription materialization or dematerialization, or during the process of refreshing a publication subscription. The subscription can be to a table replication definition, function replication definition, or publication.

  See the *Replication Server Administration Guide Volume 1* for more information about subscriptions.

- Execute check subscription at the Replication Server that manages the database where the replicate data is to be stored or the Replication Server that manages the primary database.

  The results of check subscription differ depending on where the command is executed. If the Replication Server manages both the primary and replicate database, check subscription returns two status messages.

- To check publication status, use check publication. See check publication for more information.
- Refer to the *Replication Server Troubleshooting Guide* for detailed information about monitoring subscriptions using check subscription.

Messages returned by *check subscription*

- When you execute check subscription at a replicate Replication Server, it returns one of these messages.

  In a warm standby application, there may be two lines of output showing the status at the active and at the standby replicate database.

INVALID	`sub_name doesn't exist`
REMOVING	`REMOVING subscription` *`sub_name`* `from system tables at the Replicate.`
DEMATERIALIZING	`Subscription` *`sub_name`* `is DEMATERIALIZING at the Replicate.`
VALID	`Subscription` *`sub_name`* `is VALID at the Replicate.`
VALIDATING	`Subscription` *`sub_name`* `is VALIDATING at the Replicate.`
MATERIALIZED	`Subscription` *`sub_name`* `has been MATERIALIZED at the Replicate.`
ACTIVE	`Subscription` *`sub_name`* `is ACTIVE at the Replicate.`

ACTIVATING	`Subscription` *`sub_name`* `is ACTIVATING at the Replicate.`
ACTIVATING	`Subscription` *`sub_name`* `is ACTIVATING at the Standby of the Replicate.`
QCOMPLETE and ACTIVE	`Subscription` *`sub_name`* `is ACTIVE at the Replicate and Materialization Queue has been completed.`
QCOMPLETE	`Materialization Queue for Subscription` *`sub_name`* `has been completed.`
ACTIVE and not QCOMPLETE	`Subscription` *`sub_name`* `is ACTIVE at the Replicate, but Materialization Queue for it has not been completed.`
DEFINED	`Subscription` *`sub_name`* `has been defined at the Replicate.`

- In addition to the above messages, executing check subscription at a replicate Replication Server may return one of these messages:

ERROR	`Subscription` *`sub_name`* `has experienced an unrecoverable error during Materialization or Dematerialization. Please consult the error log for more details.`
PENDING	`Other subscriptions are being created or dropped for the same replication definition/database. Subscription` *`sub_name`* `will be processed when previous requests are completed.`
RECOVERING	`Subscription` *`sub_name`* `has experienced a recoverable error during Materialization or Dematerialization. It will be recovered by Subscription Daemon (dSub).`

- When you execute check subscription at a primary Replication Server, it returns one of these messages:

INVALID	*`subscription_name`* `doesn't exist`
DEMATERIALIZING	`Subscription` *`sub_name`* `is DEMATERIALIZING at the PRIMARY.`
VALID	`Subscription` *`sub_name`* `is VALID at the PRIMARY.`
ACTIVE	`Subscription` *`sub_name`* `is ACTIVE at the PRIMARY.`
ACTIVATING	`Subscription` *`sub_name`* `is ACTIVATING at the PRIMARY.`
DEFINED	`Subscription` *`sub_name`* `has been defined at the PRIMARY.`

Permissions

Any user may execute this command.

See also

activate subscription, check publication, create subscription, define subscription, drop subscription, validate subscription

configure connection

Description	Changes the attributes of a database connection.

Note configure connection is identical in behavior to the alter connection command.

Syntax	For syntax information, see alter connection.
Usage	For usage information, see alter connection.

configure logical connection

Description	Changes attributes of a logical connection.

Note configure logical connection is identical to the alter logical connection command.

Syntax	For syntax information, see alter logical connection.
Usage	For usage information, see alter logical connection.

configure replication server

Description

Sets characteristics of the Replication Server, including network-based security. Configures ERSSD.

Syntax

configure replication server {
  set *repserver_param* to '*value*' |
  set *route_param* to '*value*' |
  set *database_param* to '*value*' |
  set *logical_database_param* to '*value*' |
  set *security_param* to '*value*' |
  set *id_security_param* to '*value*' |
  set *security_services* [to] 'default' }

Parameters

repserver_param
The name of a parameter that affects the Replication Server. Refer to Table 3-17 and Table 3-21 for a description of parameters and values.

value
A setting for a configuration parameter.

Table 3-17: Replication Server configuration parameters

repserver_param	value
cm_fadeout_time	The number of seconds of idle time before Replication Server closes a connection with the RSSD. A value of –1 specifies that a connection will never be closed. Default: 300 seconds Minimum: 1 second Maximum: 2,147,483,648 seconds
cm_max_connections	The maximum number of outgoing connections available to the connection manager. The value must be greater than 0. Default: 64
current_rssd_version	The Replication Server version supported by this RSSD. The Replication Server checks this value at startup. **Note** Do not change the value for this parameter. This value should only be modified by the rs_init program when you upgrade or downgrade. Default: N/A

repserver_param	value
deferred_queue_size	The maximum size of an Open Server deferred queue. If Open Server limits are exceeded, increase the maximum size. The value of deferred_queue_size must be greater than 0. **Note** You must restart the Replication Server for any changes to this parameter to take effect. Default: 2048 on Linux and HPIA32 1024 on other platforms
ha_failover	Enables or disables Sybase Failover support for new database connections from the Replication Server to Adaptive Servers. Values are: • on - Failover is enabled • off - Failover is disabled Default: off
id_server	The name of the ID Server for this Replication Server. **Note** Do not change the value for this parameter unless required in order to specify a different ID Server name for all Replication Servers in the replication system. You specified the ID Server name when you installed the current Replication Server using the rs_init program. Default: N/A
init_sqm_write_delay	Write delay for the Stable Queue Manager if queue is being read. Default: 1000 milliseconds
init_sqm_write_max_delay	The maximum write delay for the Stable Queue Manager if the queue is not being read. Default: 10,000 milliseconds
memory_limit	The maximum total memory the Replication Server can use. Values for several other configuration parameters are directly related to the amount of memory available from the memory pool indicated by memory_limit. These include md_sqm_write_request_limit, queue_dump_buffer_size, sqt_max_cache_size, sre_reserve, and sts_cachesize. Default: 20MB
minimum_rssd_version	The minimum version of the Replication Server that can use this RSSD. When the current_rssd_version is greater than the version of the Replication Server, this value is checked when the Replication Server is started. **Note** Do not change the value for this parameter. This value should only be modified by the rs_init program when you upgrade or downgrade. Default: N/A

repserver_param	**value**
num_client_connections	The maximum number of incoming client connections allowed. If Open Server limits are exceeded, increase the maximum number. The value must be greater than or equal to 30. Default: 30
num_concurrent_subs	The maximum number of concurrent subscription materialization/ dematerialization requests allowed. (Limit applies to atomic and non-atomic materialization only; does not apply to bulk materialization.) Requests over the maximum are fulfilled after other requests have been fulfilled. The minimum value is 1. Default: 10
num_msgqueues	The maximum number of Open Server message queues allowed. If Open Server limits are exceeded, increase the maximum number. The value must be greater than the num_threads setting. Default: 178
num_msgs	The maximum number of Open Server message queue messages allowed. If Open Server limits are exceeded, increase the maximum number. Default: 45,568
num_mutexes	The maximum number of Open Server mutexes allowed. If Open Server limits are exceeded, increase the maximum number. The value must be greater than the num_threads setting. Default: 128
num_stable_queues	The maximum number of stable queues allowed (HP9000 only). Each stable queue uses 32,768 bytes of shared memory. The minimum number of stable queues allowed is 32. Each standby database connection uses an additional 16,384 bytes of shared memory. Every two standby database connections count as one additional stable queue. Default: 32
num_threads	The maximum number of Open Server threads allowed. If Open Server limits are exceeded, increase the maximum number. The value must be greater than or equal to 20. Default: 50
oserver	The name of the current Replication Server. **Note** Do not change the value for this parameter. You specified the current Replication Server name when you installed it using rs_init. Default: N/A

repserver_param	value
password_encryption	Indicates if password encryption is enabled/disabled: • 1 – encryption enabled • 0 – encryption disabled Default: 0
prev_min_rssd_version	Following an rs_init installation upgrade, this value contains the previous value of minimum_rssd_version. **Note** Do not change the value for this parameter. This value should be modified only by rs_init when you upgrade or downgrade. Default: N/A
prev_rssd_version	Following an rs_init installation upgrade, this value contains the previous value of current_rssd_version. **Note** Do not change the value for this parameter. This value should be modified only by rs_init when you upgrade or downgrade. Default: N/A
queue_dump_buffer_size	The maximum command length, in bytes, used by the sysadmin dump_queue command. Commands larger than the specified length are truncated. The range is 1000 to 32,768. Default: 1000 bytes
rec_daemon_sleep_time	Specifies the sleep time for the recovery daemon, which handles “strict” save interval messages in warm standby applications and certain other operations. Default: 2 minutes
rssd_error_class	Error class for the RSSD. Default: rs_sqlserver_error_class
send_enc_password	Ensures that all Replication Server client connections are made with encrypted passwords—except for the first connection to the RSSD. Values are “on” and “off.” See the *Replication Server Administration Guide Volume 1* for more information. Default: off
smp_enable	Enables symmetric multiprocessing (SMP). Specifies whether Replication Server threads should be scheduled internally by Replication Server or externally by the operating system. When Replication Server threads are scheduled internally, Replication Server is restricted to one machine processor, regardless of how many may be available. Values are “on” and “off.” Default: off

repserver_param	value
sqm_recover_segs	Specifies the number of stable queue segments Replication Server scans during initialization. Default: 1 Minimum: 1 Maximum: 2,147,483,648
sqm_warning_thr1	Percent of partition segments (stable queue space) to generate a first warning. The range is 1 to 100. Default: 75
sqm_warning_thr2	Percent of partition segments used to generate a second warning. The range is 1 to 100. Default: 90
sqm_warning_thr_ind	Percent of total partition space that a single stable queue uses to generate a warning. The range is 51 to 100. Default: 70
sqm_write_flush	Specifies whether or not writes to memory buffers are flushed to the disk before the write operation completes. Values are “on” and “off.” Default: on
sqt_init_read_delay	The length of time an SQT thread sleeps while waiting for an SQM read before checking to see if it has been given new instructions in its command queue. With each expiration, if the command queue is empty, SQT doubles its sleep time up to the value set for sqt_max_read_delay. Default: 1 ms Minimum: 0 ms Maximum: 86,400,000 ms (24 hours)
sqt_max_cache_size	Maximum SQT (Stable Queue Transaction) interface cache memory, in bytes. Default: 1,048,576 bytes
sqt_max_read_delay	The maximum length of time an SQT thread sleeps while waiting for an SQM read before checking to see if it has been given new instructions in its command queue. Default: 1 ms Minimum: 0 ms Maximum: 86,400,000 ms (24 hours)
sre_reserve	The amount of additional space to allocate for new subscriptions. For example, 100 (100%) means double the current space. The range is 0 to 500. To update the sre_reserve parameter for a replication definition, insert into or update the rs_config system table directly. Default: 0

repserver_param	value
stats_reset_rssd	Indicates whether RSSD truncates previous sampling data or overwrites it with new information. Values: On – keep previous sampling data. Off – overwrite old sampling data with new information. Default: on
stats_sampling	Enables sampling counters. Default: off
stats_show_zero_counters	Specifies whether the admin stats command reports counters with zero observations for a specified sample period. The values are: • "on" – counters with zero observations are reported. • "off" – counters with zero observations are not reported. Default: off
sts_cachesize	The total number of rows cached for each cached RSSD system table. Increasing this number to the number of active replication definitions prevents Replication Server from executing expensive table lookups. Default: 100
sts_full_cache_ *system_table_name*	Specifies an RSSD system table that is to be fully cached.
sub_daemon_sleep_time	Number of seconds the subscription daemon sleeps before waking up to recover subscriptions. The range is 1 to 31,536,000. Default: 120 seconds
varchar_truncation	Enables truncation of varchar columns at the primary or replicate Replication Server. Set varchar_truncation at the replicate Replication Server when a character set conversion takes place at both Replication Servers. Default: off

route_param
: Affects routes. See Table 3-16 on page 147 for a list and description of route parameters. configure replication server sets parameter values for all routes that originate at the source Replication Server.

database_param
: Affects connections. See Table 3-14 on page 109 for a list and description of connection parameters. configure replication server sets parameter values for all connections that originate at the source Replication Server.

logical_database_param

Affects logical connections. See Table 3-16 on page 147 for a list and description of parameters. configure replication server sets parameter values for all logical connections that originate at the source Replication Server

security_param

Affects network-based security. See Table 3-18 on page 172 for a list and description of parameters.

Table 3-18: Parameters affecting network-based security

security_param	value
msg_confidentiality	Indicates whether Replication Server sends and receives encrypted data. If set to "required," outgoing data is encrypted. If set to "not required," Replication Server accepts incoming data that is encrypted or not encrypted. Default: not_required
msg_integrity	Indicates whether data is checked for tampering. Default: not_required
msg_origin_check	Indicates whether the source of data should be verified. Default: not_required
msg_replay_detection	Indicates whether data should be checked to make sure it has not been intercepted and resent. Default: not_required
msg_sequence_check	Indicates whether data should be checked to make sure it was received in the order sent. Default: not_required
mutual_auth	Indicates whether the remote server must provide proof of identify before a connection is established. Default: not_required
security_mechanism	The name of the third-party security mechanism enabled for the pathway. Default: first mechanism listed in the SECURITY section of *libtcl.cfg*
send_enc_password	Ensures that all Replication Server client connections are made with encrypted passwords—except for the first connection to the RSSD. Values are "on" and "off." Default: off
unified_login	Indicates how Replication Server seeks to log in to remote data servers and accepts incoming logins. The values are: • "required" – always seeks to log in to remote server with a credential. • "not_required" – always seeks to log in to remote server with a password. Default: not_required

security_param	value
use_security_services	Tells Replication Server whether to use security services. If use_security_services is “off,” no security features take effect. **Note** This parameter can only be set by configure replication server.
use_ssl	Indicates whether Replication Server is enabled for session-based SSL security. The values are: • “on” – Replication Server is enabled for SSL. • “off” – Replication Server is not enabled for SSL. Default: off

id_security_param
Affects network-based security for the ID Server. See Table 3-19 on page 173 for a list and description of these parameters.

Table 3-19: Security parameters for connecting to the ID Server

security_param	value
id_msg_confidentiality	Indicates whether Replication Server sends and receives encrypted data packets. If set to “required,” outgoing data is encrypted. If set to “not required,” Replication Server accepts incoming data that is encrypted or not encrypted. Default: not required
id_msg_integrity	Indicates whether data packets are checked for tampering. Default: not required
id_msg_origin_check	Indicates whether the source of data packets should be verified. Default: not required
id_msg_replay_detectionr	Indicates whether data packets should be checked to make sure they have not been intercepted and resent. Default: not required
id_ msg_sequence_check	Indicates whether data packets should be checked to make sure they are received in the order sent. Default: not required
id_mutual_auth	Requires the ID Server to provide proof of identify before Replication Server establishes a connection. Default: not required
id_security_mech	Specifies the name of the supported security mechanism. Supported security mechanisms are listed under SECURITY in the *libtcl.cfg* file. If no name is specified, Replication Server uses the default mechanism. Default: the first mechanism in the list

security_param	value
id_unified_login	Indicates how Replication Server seeks to connect to ID Server. The values are: required – always seeks to log in to ID Server with a credential. not required – always seeks to log in to ID Server with a password. **Note** Only the “sa” user can log in to Replication Server without a credential if unified_login is “required.” If the security mechanism should fail, the “sa” user can log in and disable unified_login. Default: not required

set security_services [to] 'default'

Resets all network-based security features for the connection to match the global settings of your Replication Server. It does not reset the use_security_services feature.

If Replication Server supports more than one security mechanism, set security_services [to] 'default' also sets the security mechanism to the default, the first mechanism listed in the SECURITY section of the *libtcl.cfg* file.

Examples

Example 1 Sets Replication Server to send data in encrypted format:

```
configure replication server
 set id_msg_confidentiality to 'required'
```

Example 2 Sets all security features to match the global settings:

```
configure replication server
 set security_services to 'default'
```

Example 3 Changes the rsi_save_interval parameter to two minutes for all routes originating at the current Replication Server:

```
suspend route to each_dest_replication_server

 configure replication server
 set rsi_save_interval to '2'

 resume route to each_dest_replication_server
```

Usage

- Each parameter has two values: the configured value and the run value. Replication Server uses the configured value when it restarts. The run value is the value the Replication Server is using currently. When you start Replication Server the values are equal.
- Configured values are stored in the rs_config system table in the RSSD. For a description of the table, see rs_config in Chapter 8, “Replication Server System Tables.”

- varchar_truncation enables truncation of varchar columns at the primary or replicate Replication Server. When incoming varchar data exceeds the column length specified in the replication definition, the following occurs:

Table 3-20: varchar_truncation

	varchar_truncation set at primary Replication Server	**varchar_truncation set at replicate Replication Server**
varchar_trunction set to "on"	Replication Server truncates incoming data to the length specified in the replication definition.	Replication Server truncates incoming data to the length specified in the replication definition.
varchar_truncation set to "off"	RepAgent prints a message in the Replication Server log, and Replication Server ignores rows that exceed the column length specified in the replication definition.	Replication Server prints a message in the Replication Server log, and the DSI shuts down.

- Use ha_failover to enable Sybase failover support. In the event of an ASE server failover, all connections from Replication Server to ASE will fail. Replication Server will retry connections. Setting ha_failover to on will allow the new connections to failover to the new ASE server.
- Use ERSSD configuration parameters to configure backup time, directory location and RepAgent name.

Table 3-21: ERSSD configuration parameters

ERSSD configuration parameter	**Value**	**Default**
erssd_backup_start_time	Time the backup starts. Specified as: "hh:mm AM" or "hh:mm PM", using a 12-hour clock, or "hh:mm" using a 24-hour clock.	Default: 01:00 AM
erssd_backup_start_date	Date the backup begins. Specified as "MM/DD/YYYY".	Default: current date
erssd_backup_interval	Interval between backups of database and log. Specified as "nn hours" or "nn minutes" or "nn seconds".	Default: 24 hours
erssd_backup_path	Location of stored backup files. Should be a full directory path. Configuring this path causes immediate backup.	Default: Same directory as the transaction log mirror; initial value specified in rs_init.

ERSSD configuration parameter	Value	Default
erssd_ra	Configures Replication Agent name, in order to create a route from the current site to another Replication Server. This server name must exist in the interfaces name.	*erssd_name*_ra

Replication Server parameters

- Replication Server parameters specify default values that affect the local Replication Server.
- Replication Server parameters are static. You must restart Replication Server for them to take effect.

Route parameters

- Route parameters specify default values for all routes that originate at the source Replication Server.
- You can override default values specified using configure replication server by using alter route to set values for individual routes.
- Route parameters are semi-dynamic. You must suspend all routes originating at the current Replication Server before executing the configure replication server command. After you have changed the parameter, you must resume all routes for the change to take effect.

Database parameters

- Database parameters specify default values for all connections that originate at the source Replication Server.
- You can override default values specified using configure replication server by using alter connection to set values for an individual connection.
- Database parameters are semi-dynamic. You must suspend all connections originating at the current Replication Server before executing configure replication server. After you change the parameter, resume all connections for the change to take effect.

Logical database parameters

- Logical database parameters specify default values for logical connections that originate at the source Replication Server.
- You can override default values specified using configure replication server by using configure logical connection to set values for a specific logical connection.

- Logical database parameters are dynamic. They take effect immediately.

Network-based security parameters

- With the exception of use_security_services and use_ssl, security parameters configured with configure replication server are dynamic; they take effect immediately.
- use_security_services and use_ssl are static. If you change their values, you must restart Replication Server for the change to take effect.
- Default network-based security parameters set with configure replication server specify values for all incoming and outgoing pathways related to the current Replication Server.
- You can override default security settings specified using configure replication server by using alter route or alter connection to reset security values for individual *outgoing* pathways.
- If unified_login is set to "required," *only* the "sa" user can log in to the Replication Server without a credential. If the security mechanism should go down, the "sa" user can log in to Replication Server with a password and disable unified_login.
- A Replication Server can support more than one security mechanism. Each supported mechanism is listed in the *libtcl.cfg* file under SECURITY.
- Both ends of a route must use compatible Security Control Layer (SCL) drivers with the same security mechanisms and security settings. It is the replication system Administrator's responsibility to choose and set security features for each server. Replication Server does not query the security features of remote servers before it attempts to establish a connection. Network connections fail if security features at both ends of the pathway are not compatible.
- Message encryption is a costly process with severe performance penalties. In most instances, it is wise to set msg_confidentiality to "required" only for certain pathways. Alternatively, choose a less costly feature, such as msg_integrity, to ensure security.

Permissions

configure replication server requires "sa" permission.

See also

admin security_property, admin security_setting, alter connection, alter route, configure connection, configure route, create connection, create route, set proxy

configure route

Description

Changes the attributes of a route from the current Replication Server to a remote Replication Server.

Note configure route is identical to the alter route command.

Syntax

See alter route for syntax information.

Usage

See alter route for usage information.

create article

Description

Creates an article for a table or function replication definition and specifies the publication that is to contain the article.

Syntax

```
create article article_name
          for pub_name
with primary at data_server.database
with replication definition {table_rep_def | function_rep_def}
          [where {column_name | @param_name}
               {< | > | >= | <= | = | &} value
          [and {column_name | @param_name}
               {< | > | >= | <= | = | &} value]...
          [or where {column_name | @param_name}
               {< | > | >= | <= | = | &} value
          [and {column_name | @param_name}
               {< | > | >= | <= | = | &} value]...]...]
```

Parameters

article_name
A name for the article. It must conform to the rules for identifiers and be unique within the publication.

for pub_name
The name of the publication that contains the article.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

with replication definition table_rep_def
Specifies the name of the table replication definition the article is for.

with replication definition function_rep_def
Specifies the name of the function replication definition the article is for.

where

Sets criteria for the column or parameter values to be replicated via a subscription to the publication that contains this article. If no where clause is included, all rows or parameters are replicated.

A where clause is composed of one or more simple comparisons, where a searchable column or searchable parameter is compared to a literal value with one of the following relational operators: <, >, <=, >=, =, or &. (The & operator is supported *only* for rs_address columns or parameters.) You can join comparisons with the keyword and.

Column or parameter names used in a where clause must also be included in the searchable columns list of the table replication definition or the searchable parameters list of the function replication definition.

You can include multiple where clauses in an article, separated with the keyword or.

The maximum size of a where clause in an article is 255 characters.

column_name

A column name from the primary table, for an article that contains a table replication definition.

@param_name

A parameter name from a replicated stored procedure, for an article that contains a function replication definition.

value

A value for a specified column or parameter. See "Datatypes" on page 21 for entry formats for values for different datatypes.

Column or parameter names used in the expression must be included in the searchable columns or searchable parameters list of the replication definition.

Examples

Example 1 Creates an article called titles_art for the publication pubs2_pub, based on the replication definition titles_rep:

```
create article titles_art
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition titles_rep
```

Example 2 Creates an article called titles_art for the publication pubs2_pub, as in the previous example. This command includes a where clause that replicates only the rows for popular computing books, for which the type column is set to "popular_comp":

```
create article titles_art
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition titles_rep
 where type = 'popular_comp'
```

Example 3 Creates an article called titles_art for the publication pubs2_pub, as in the previous examples. This command includes two where clauses that together replicate the rows for both popular computing books and traditional cookbooks:

```
create article titles_art
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition titles_rep
 where type = 'popular_comp'
 or where type = 'trad_cook'
```

Usage

- Use create article to specify a replication definition for which you want to replicate data using a specified publication. Optional where clauses help determine which data is replicated.
- Execute create article at the Replication Server that manages the database where the primary data is stored.
- Using create article automatically invalidates the publication the article is for. You cannot create new subscriptions until you validate the publication. You cannot replicate data for the new articles until you refresh the subscription.
- For more information about working with replication definitions, articles, and publications, see the *Replication Server Administration Guide Volume 1*.

  For more information about subscribing to publications, refer to Chapter 10, "Managing Subscriptions," in the same book.
- Replication Server distributes information about a publication and its articles to a replicate site only when you create or refresh a subscription for the publication.

Requirements for using *create article*

- Before executing create article, make sure that:
  - The publication for which you are creating the article already exists.
  - The replication definition for the article already exists.

- A Replication Server of release 11.0.*x* does not receive information about publications or articles and cannot subscribe to them.

Adding articles to a new publication

- After you create a publication, you use create article to create articles and assign them to the publication. An article specifies a table replication definition or function replication definition and a parent publication. Optionally, it may also include where clauses according to the needs of the subscribing replicate site.

  A publication must contain at least one article before it can be validated and before you can create subscriptions for it. See create publication for more information.

Articles and subscriptions

- When you create a subscription for a publication, Replication Server creates an internal subscription for each of its articles.
- Including multiple where clauses for an article, separated by the or keyword, allows you to work around the Replication Server restriction that allows only one where clause per subscription. A publication subscription cannot include a where clause—use where clauses in the articles instead.

Adding articles to a publication with a subscription

- If you add a new article to an existing publication, or drop an article from the publication, the publication is invalidated. Although replication for existing articles continues unaffected, in order to begin replication for the new articles you must:
  - Validate the publication when you finish making changes to the publication, then
  - Refresh the publication subscription.

  See create subscription and define subscription for more information on the two methods of refreshing publication subscriptions. See also validate publication.

Permissions

create article requires "create object" permission.

See also

check publication, create function replication definition, create publication, create replication definition, create subscription, define subscription, drop article, drop publication, validate publication

create connection

Description

Adds a database to the replication system and sets configuration parameters for the connection. To create a connection for an Adaptive Server database, use Sybase Central or rs_init.

Syntax

```
create connection to data_server.database
set error class [to] error_class
set function string class [to] function_class
set username [to] user
[set password [to] passwd]
[set database_param [to] 'value' [set database_param [to] 'value']...]
[set security_param [to] 'value' [set security_param [to] 'value']...]
[with {log transfer on, dsi_suspended}]
[as active for logical_ds.logical_db |
as standby for logical_ds.logical_db
[use dump marker]]
```

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.

error_class
The error class that is to handle errors for the database.

function_class
The function string class to be used for operations in the database.

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.

passwd
The password for the maintenance user login name. You *must* specify a password unless a network-based security mechanism is enabled.

database_param
A parameter that affects database connections from the Replication Server. Parameters and values are described in Table 3-14 on page 109.

value
A character string that contains a value for the option.

security_param
A parameter that affects network-based security. See Table 3-27 on page 240 for a list and description of security parameters that you can set with create connection.

log transfer on
Indicates that the connection may be a primary data source or the source of replicated functions. When the clause is present, Replication Server creates an inbound queue and prepared to accept a RepAgent connection for the database. If you omit this option, the connection cannot accept input from a RepAgent.

dsi_suspended
Starts the connection with the DSI thread suspended. You can resume the DSI later. This option is useful if you are connecting to a non-Sybase data server that does not support Replication Server connections.

as active for
Indicates that the connection is a physical connection to the active database for a logical connection.

as standby for
Indicates that the connection is a physical connection to the standby database for a logical connection.

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

use dump marker
Tells Replication Server to apply transactions to a standby database after it receives the first dump marker after the enable replication marker in the transaction stream from the active database. Without this option, Replication Server applies transactions it receives after the enable replication marker.

Examples

Example 1 Creates a connection for the pubs2 database in the SYDNEY_DS data server. Replication Server will use the ansi_error error class to handle errors for the database. It will use the function strings in the sqlserver_derived_class function string class for data manipulation operations. The connection will use the pubs2_maint login name with the password pubs2_maint_ps to log into the pubs2 database:

```
create connection to SYDNEY_DS.pubs2
 set error class ansi_error
 set function string class sqlserver_derived_class
 set username pubs2_maint
 set password pubs2_maint_pw
```

Example 2 Creates a connection similar to the first example, but with log transfer specified. This allows the connection to accept input from a RepAgent. The connection is with a database that contains primary data or that will be a source of replicated functions:

```
create connection to TOKYO_DS.pubs2
 set error class ansi_error
 set function string class sqlserver_derived_class
 set username pubs2_maint
 set password pubs2_maint_pw
 with log transfer on
```

Usage

- Use create connection to add a database to the replication system. Normally, you use this command to add connections to non-Sybase databases. To create a standard connection with an Adaptive Server database, use Sybase Central or rs_init.
- To create a connection that uses heterogeneous datatype support (HDS) to translate datatypes from the primary to the replicate database, you can also use scripts provided by Sybase that both create the connection and install HDS. See the *Replication Server Configuration Guide* for your platform for instructions.
- Execute create connection at the Replication Server that manages the database.
- Replication Server distributes database connection information to qualifying sites through out the replication system. The changes do not appear immediately at all sites because of normal replication system lag time.
- You must specify an error class, even if you use the default error class: rs_sqlserver_error_class.
- Only one connection is allowed per database. This is enforced by the ID Server, which registers each database in its rs_idnames system table. The ID Server must be available when you create a connection for a database.
- Use set function string class [to] *function_class* to activate class-level translations for non-Sybase data servers.

Database connection parameters

- Replication Server configuration parameters are stored in the rs_config system table. See the *Replication Server Administration Guide Volume 1* for more information about the database connection parameters in the rs_config system table.
- See the *Replication Server Administration Guide Volume 2* for more information about configuring parallel DSI threads.
- Use assign action to enable retry of transactions that fail due to specific data server errors.

The *dump_load* configuration parameter

- Before setting dump_load to "on," create function strings for the rs_dumpdb and rs_dumptran functions. Replication Server does not generate function strings for these functions in the system-provided classes or in derived classes that inherit from these classes.

The *save_interval* configuration parameter

- Set save_interval to save transactions in the DSI queue that can be used to resynchronize a database after it has been restored from backups. Setting a save interval is also useful when you set up a warm standby of a database that holds replicate data or receives replicated functions. You can use sysadmin restore_dsi_saved_segments to restore backlogged transactions.

Error classes and function classes

- Table 3-22 shows the error and function classes that Replication Server provides for database connections.

Table 3-22: Adaptive Server error and function classes

Class name	Description
rs_sqlserver_error_class	Error action assignments for Adaptive Server databases.
rs_sqlserver_function_class	Function-string class for Adaptive Server databases. Cannot participate in function string inheritance. Replication Server generates function strings automatically.
rs_default_function_class	Function-string class for Adaptive Server databases. You cannot modify function strings. You can specify this class as a parent class, but not as a derived class. Replication Server generates function strings automatically.
rs_db2_function_class	Function-string class for DB2 databases. You cannot modify function strings. You can specify this class as a parent class, but not as a derived class. Replication Server generates function strings automatically.
rs_informix_function_class	Function-string class for Informix databases. You cannot modify function strings. You can specify this class as a parent class, but its derived classes cannot inherit any class-level translations from the parent class. Replication Server generates function strings automatically.

Class name	Description
rs_ms_function_class	Function-string class for Microsoft SQL Server databases. You cannot modify function strings. You can specify this class as a parent class, but its derived classes cannot inherit any class-level translations from the parent class. Replication Server generates function strings automatically.
rs_oracle_function_class	Function-string class for Oracle databases. You cannot modify function strings. You can specify this class as a parent class, but its derived classes cannot inherit any class-level translations from the parent class. Replication Server generates function strings automatically.
rs_udb_function_class	Function-string class for UDB databases. You cannot modify function strings. You can specify this class as a parent class, but its derived classes cannot inherit any class-level translations from the parent class. Replication Server generates function strings automatically.

Note The rs_dumpdb and rs_dumptran system functions are not initially defined, even for function-string classes in which Replication Server generates default function strings. If you intend to use coordinated dumps, you must create function strings for these functions. Note also that you cannot perform coordinated dumps on a standby database. See the *Replication Server Administration Guide Volume 2* for more information about using function strings. For more information about the rs_dumpdb and rs_dumptran functions, see Chapter 4, "Replication Server System Functions."

User name and password

- You specify the maintenance user login name and password when creating the connection. The maintenance user login name must be granted all necessary permissions to maintain replicated data in the database.

Note When two sites in a replication system have the same database name, the maintenance user login names must be different. The default login name, created by Sybase Central or rs_init is *DB_name*_maint. When setting up the system, change one of the login names so each are unique.

Warm standby applications

- To create a logical connection for a warm standby application, use create logical connection.
- In a warm standby application, the connections for the active database and the standby database must have log transfer on.

- The function-string class for a database in a warm standby application is used only when the database is the active database. Replication Server uses rs_default_function_class for the standby database.

Changing connection attributes

- Use alter connection to change the attributes of a connection.
- If the password of the maintenance user has been changed, use alter connection to enter the new password.

Network-based security parameters

- Both ends of a connection must use compatible Security Control Layer (SCL) drivers with the same security mechanisms and security features. The remote server must support the set proxy or equivalent command. It is the replication system Administrator's responsibility to choose and set security features for each server. The Replication Server does not query the security features of remote servers before attempting to establish a connection. Connections fail if security features at both ends of the connection are not compatible.
- create connection specifies security settings for an outgoing connection from Replication Server to a target data server. Security features set by create connection override those set by configure replication server.
- If unified_login is set to "required," *only* the replication system Administrator with "sa" permission can log in to the Replication Server without a credential. If the security mechanism should fail, the replication system Administrator can log in to Replication Server with a password and disable unified_login.
- A Replication Server can have more than one security mechanism; each supported mechanism is listed in the *libtcl.cfg* file under SECURITY.
- Message encryption is a costly process with severe performance penalties. In most instances, it is wise to set msg_confidentiality to "required" only for certain connections. Alternatively, choose a less costly security feature, such as msg_integrity.

Permissions

create connection requires "sa" permission.

See also

alter connection, configure connection, configure connection, create error class, create function string class, create logical connection, alter route, drop connection, resume connection, suspend connection

create database replication definition

Description

Creates a replication definition for replicating a database or a database object.

Syntax

```
create database replication definition db_repdef
        with primary at server_name.db
        [ [ not ] replicate DDL ]
        [ [ not ] replicate setname setcont ]
        [ [ not ] replicate setname setcont ]
        [ [ not ] replicate setname setcont ]
        [ [ not ] replicate setname setcont ]

setname  ::= { tables | functions | transactions | system procedures }

setcont  ::= [ in ( [ owner1.]name1 [, [owner2.]name2 [, ... ] ] ) ]
```

Parameters

db_repdef
Name of the database replication definition.

server_name.db
Name of the primary server/database combination. For example: TOKYO.dbase.

[not] replicate DDL
Tells Replication Server whether or not to send DDL to subscribing databases. If “replicate DDL” is not included, or the clause includes “not,” DDL is not sent to the replicate database.

[not] replicate *setname setcont*
Tells Replication Server whether or not to send objects in the *setname* category to the replicate database.

If you omit the system procedures *setname* or include the not option, Replication Server does not replicate the system procedures.

If you omit replicate tables, functions, or transactions *setname* or include the not option, Replication Server replicates all objects of the *setname* category.

owner
An owner of a table or a user who executes a transaction. Replication Server does not process owner information for functions or system procedures.

You can replace *owner* with a space surrounded by single quotes or with an asterisk.

- A space (‘ ‘) – indicates no owner.
- An asterisk (*) – indicates all owners. Thus, for example, *.publisher means all tables named publisher, regardless of owner.

name

The name of a table, function, transaction, or system procedure.

You can replace *name* with a space surrounded by single quotes or with an asterisk.

- A space (‘ ‘) – indicates no name. For example, maintuser.’ ‘ means all unnamed maintenance user transactions.
- An asterisk (*) – indicates all names. Thus, for example, robert.* means all tables (or transactions) owned by robert.

Examples

Example 1 Creates a database replication definition rep_1B. This database replication definition specifies that only tables employee and employee_address are replicated:

```
create database replication definition rep_1B
  with primary at PDS.pdb
  replicate tables in (employee, employee_address)
```

Example 2 Creates a database replication definition rep_2. In this example, the database my_db is replicated, DDL is replicated, but system procedures are not replicated:

```
create database replication definition rep_2
  with primary at dsA.my_db
  replicate DDL
  not replicate system procedures
```

Usage

- create database replication definition lets you replicate all, all with some exceptions, or only some of, the tables, functions, transactions, and system procedures from the primary database.
- Use create database replication definition alone or in conjunction with table and function replication definitions.
- With only a database replication definition, that is, without table or function replication definitions, Replication Server cannot transform data. However, it can perform minimal column replication. This data replication behavior is similar to that of a default warm standby.

For a database replication definition to replicate encrypted columns without using a table level replication definition, you must define the encryption key for the encrypted column with INIT_VECTOR NULL and PAD NULL. If a table in the database includes encrypted columns where the encryption key was created with random padding (the default) or initialization vectors, a table level replication definition is required to ensure database consistency.

- Database replication definitions are global objects. They are replicated to all Replication Servers that have a route from the defining Replication Server.
- Database replication definitions do not affect request function replication.
- Table and function filters are not implemented if table and function subscriptions exist.
- Replication Server does not process owner information for functions and system procedures.

Owner information

- Replication Server always uses owner information provided in the database replication definition.
- Replication Server does not use owner information provided in a table replication definition if the table is marked with sp_reptostandby.
- Replication Server only uses owner information provided in a table replication definition if the table is marked by sp_setreptable with the owner_on clause.

See also

alter database replication definition, drop database replication definition

create error class

Description

Creates an error class.

Syntax

create error class *error_class*

Parameters

error_class
The name for the new error class. The name must be unique in the replication system and must conform to the rules for identifiers.

Examples

This example creates a new error class named pubs2_db_err_class:

```
create error class pubs2_db_err_class
```

Usage

- Use create error class to create an error class. An error class is a name used to group error action assignments for a database.
- This command has the following requirements:
  - Routes must exist from the Replication Server where an error class is created to the Replication Servers managing data servers that are to use the error class.
  - The rs_sqlserver_error_class is the default error class provided for Adaptive Server databases. Initially, this error class does not have a primary site. You must create this error class at a primary site before you can change default error actions.
  - After using create error class, use the rs_init_erroractions stored procedure to initialize the error class.
- Associate an error class with a database using create connection or alter connection. Each database can have one error class. An error class can be associated with multiple databases.
- Replication Server distributes the new error class to qualifying sites through out the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

Assigning error actions

- Use assign action to change the Replication Server response to specific data server errors. Actions are assigned at the Replication Server where the error class is created.

Dropping error classes

- Use drop error class to remove an error class and any actions associated with it.

Permissions

create error class requires “sa” permission.

See also

alter connection, assign action, create connection, drop error class, rs_init_erroractions

create function

Description

Creates a user-defined function.

Note You do not need to create user-defined functions for use with replicated functions. When you create a function replication definition, a user-defined function is created automatically. For more information, see create function replication definition.

If your application uses asynchronous procedure delivery associated with table replication definitions, you may need to create user-defined functions. For more information, see the *Replication Server Administration Guide Volume 2*.

Syntax

create function *replication_definition.function*
([*@param_name datatype* [, *@param_name datatype*]...])

Parameters

replication_definition
The name of the replication definition the function is for. You can create only one user-defined function for all the replication definitions for the same table. If there are multiple replication definitions for the same table, you can specify the name of any one of them. However, each replication definition has its own function string for the user-defined function.

function
The name of the function. The name must be unique for the replication definition and must conform to the rules for identifiers. The names of the system functions listed in Chapter 4, “Replication Server System Functions,”and all function names that begin with “rs_”, are reserved.

@param_name
The name of an argument to the user-defined function. Each parameter name must be preceded by an @ sign and must conform to the rules for identifiers. The value of each parameter is supplied when the function is executed.

datatype
The datatype of the parameter. Some datatypes require a length, in parentheses, after the datatype name. See “Datatypes” on page 21 for a description of the datatypes and their syntax. The datatype cannot be text, unitext, rawobject, or image.

Examples

Creates a user-defined function named newpublishers, with four parameters, for the publishers_rep replication definition:

```
create function publishers_rep.newpublishers
 (@pub_id char(4), @pub_name varchar(40),
 @city varchar(20), @state char(2))
```

Usage

- Use create function to create user-defined functions.
- Execute create function on the Replication Server where the replication definition was created.
- User-defined functions may be used in asynchronous procedure delivery. See the *Replication Server Administration Guide Volume 2* for more information about asynchronous procedures.
- You must include the parentheses () surrounding the listed parameters, even when you are defining functions with no parameters.
- For each of the three system-provided function-string classes in which the user-defined function will be used, and for each derived class that inherits from these classes, Replication Server generates a default function string for the user-defined function.
- You can customize the function string in rs_sqlserver_function_class and in user-created function-string classes using create function string.
- For each user-created base function-string class in which the user-defined function will be used, and for each derived class that inherits from such a class, you must create a function string, using create function string. The function string should invoke a stored procedure or RPC, with language appropriate for the replicate data server.
- For an overview of function-string classes, function strings, and functions, see the *Replication Server Administration Guide Volume 2*.
- Replication Server distributes the new user-defined function to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.
- When you create a user-defined function for a replication definition, it is created for all replication definitions in the primary table.

Permissions

create function requires “create object” permission.

See also

create function replication definition, create function string, drop function

create function replication definition

Description

Creates a function replication definition and user-defined function for a stored procedure that is to be replicated.

Syntax

```
create function replication definition
          function_rep_def
with primary at data_server.database
[deliver as 'proc_name']
          ([@param_name datatype [, @param_name datatype]...])
          [searchable parameters (@param_name
           [, @param_name]...)]
          [send standby {all | replication definition}
          parameters]
```

Parameters

function_rep_def

A name for the function replication definition. It must conform to the rules for identifiers.

with primary at

Specifies the data server and database containing the primary data.

data_server

The name of the data server containing the primary data. If the primary database is part of a warm standby application, *data_server* is the logical data server name.

database

The name of the database containing the primary data. If the primary database is part of a warm standby application, *database* is the logical database name.

deliver as

Specifies the name of the stored procedure to execute at the database where you are delivering the replicated function. *proc_name* is a character string of up to 200 characters. If you do not use this clause, the function is delivered as a stored procedure with the same name as the function replication definition.

@param_name

A parameter name from the function. A parameter name must not appear more than once in each clause in which it appears. You are not required to include parameters and their datatypes, but you *must* include the parentheses ( ) for this clause, whether or not you include any parameters.

datatype
The datatype of a parameter in the function. See "Datatypes" on page 21for a list of the datatypes and their syntax. Adaptive Server stored procedures and function replication definitions cannot contain parameters with the text, unitext, rawobject, and image datatypes.

searchable parameters
Specifies a list of parameters that can be used in where clauses of define subscription, create subscription, or create article. You *must* include the parentheses ( ) if you include this clause.

send standby
In a warm standby application, specifies whether to send all parameters in the function (send standby all parameters) or just those specified in the replication definition (send standby replication definition parameters) to a standby database. The default is send standby all parameters.

Examples

Example 1 Creates a function replication definition named titles_frep for a function and stored procedure of the same name. The primary data is in the pubs2 database in the LDS data server. Use a function replication definition like this for an applied function:

```
create function replication definition titles_frep
 with primary at LDS.pubs2
 (@title_id varchar(6), @title varchar(80),
  @type char(12), @pub_id char(4),
  @price money, @advance money,
  @total_sales int)
  searchable parameters (@title_id, @title)
```

Example 2 Creates a function replication definition named titles_frep for a function and stored procedure of the same name, as in the previous example. In this case, the stored procedure to be invoked in the destination database is named upd_titles. Use a function replication definition like this for a request function:

```
create function replication definition titles_frep
 with primary at LDS.pubs2
 deliver as 'upd_titles'
 (@title_id varchar(6), @title varchar(80),
  @type char(12), @pub_id char(4),
  @price money, @advance money,
  @total_sales int)
  searchable parameters (@title_id, @title)
```

Usage

- Use create function replication definition to describe a stored procedure that is to be replicated. For an overview of replicated stored procedures, see the *Replication Server Administration Guide Volume 1*.
- Execute create function replication definition at the Replication Server that manages the database where the primary data is stored.
- You can create only one function replication definition per replicated stored procedure.
- Before executing this command, be sure that:
  - The function replication definition name is unique in the replication system. Replication Server cannot always enforce this requirement when you use create function replication definition.
  - A connection exists from the Replication Server to the database where the primary data is stored. See create connection for more information. You can also create connections using rs_init. Refer to the Replication Server installation and configuration guides for your platform.
  - The name, parameters, and datatypes you specify for the function replication definition match those of the stored procedure involved. You can specify only those parameters you are interested in replicating.
- Unlike replicated stored procedures associated with table replication definitions, stored procedures associated with function replication definitions are not required to update a table. This allows you to replicate transactions that are not associated with replicated data. For more information about stored procedures, see Chapter 6, “Adaptive Server Stored Procedures.”

  See sp_setrepproc on page 442 for more information on the two types of replicated stored procedures.
- Replication Server distributes the new function replication definition to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

User-defined functions and function strings

- When you create a function replication definition, Replication Server automatically creates a corresponding user-defined function.

- For each of the three system-provided function-string classes in which the user-defined function associated with this function replication definition will be used, and for each derived class that inherits from these classes, Replication Server generates a default function string for the user-defined function.
- You can customize the function string in rs_sqlserver_function_class and in user-created function-string classes using create function string.
- For each user-created base function-string class in which the user-defined function will be used, and for each derived class that inherits from such a class, you must create a function string, using create function string. The function string should invoke a stored procedure or RPC, with language appropriate for the replicate data server.
- For an overview of function-string classes, function strings, and functions, see the *Replication Server Administration Guide Volume 2*.

The *with primary at* clause

- Use the with primary at clause to specify the data server and database containing the primary data. This is not necessarily the database that contains the invoked stored procedure.

  For applied functions (primary-to-replicate function replication) and request functions (replicate-to-primary function replication), create the function replication definition at the Replication Server managing the primary data, and specify the primary database using the with primary at clause.

The *deliver as* clause

- Use the optional deliver as clause to specify the name of the stored procedure to execute at the destination database where you are delivering the replicated function. If you do not use this clause when you create or alter the function replication definition, the function is delivered as a stored procedure with the same name as the function replication definition.

  In a warm standby database the stored procedure has the same name as in the active database so the deliver as clause is ignored.

  Typically, you would use the deliver as clause for request function delivery; that is, when a function is replicated from a replicate Replication Server to a primary Replication Server. This way, the name of the replicated function is not the same as the stored procedure that is executed.

Use this method with “round-trip” stored procedure replication, where the primary Replication Server that is the destination for the request function executes an applied function, to which the originating replicate Replication Server in turn subscribes.

See the *Replication Server Administration Guide Volume 1* for more information.

Function replication definitions for HDS parameters

- Although you cannot create function replication definitions that alter the datatype of a parameters value, you can use HDS datatype definitions to declare parameters for applied function replication definitions. Such parameters are then subject to class-level translations. See the *Replication Server Administration Guide Volume 1* for more information about HDS.
- Replication Server does not perform translations on parameter values for request functions. Note, however, that during function-string mapping Replication Server uses the delimiters defined for the parameter values of their declared datatype to generate SQL.

Altering function replication definitions

- Use alter function replication definition to add parameters or searchable parameters to an existing function replication definition. You can also specify a new stored procedure name to use when delivering the replicated function at the destination database.
- If you need to remove or rename parameters in function replication definition, you must drop all subscriptions to the function replication definition (applied functions only). Then drop the function replication definition and re-create it.

Subscribing to function replication definitions

- In order to subscribe to a function replication definition, use create subscription with the without materialization clause, or use define subscription and the other commands involving bulk materialization.

Function replication definitions and table replication definitions

- In replicating stored procedures via applied functions, it may be advisable to create table replication definitions and subscriptions for the same tables that the replicated stored procedures will affect. By doing this you can ensure that any normal transactions that affect the tables will be replicated as well as the stored procedure executions.

However, DML inside stored procedures marked as replicated is not replicated. Thus, in this case, you must subscribe to the stored procedure even if you also subscribe to the table.

- If you plan to use both kinds of replication definition for the same table, you can materialize the table data with the subscription for the table replication definition. Then you can create the subscription for the function replication definition using create subscription with the without materialization clause.

Permissions

create function replication definition requires "create object" permission.

See also

alter function replication definition, alter function string, create connection, create function string, define subscription, drop function replication definition, sp_setrepproc

create function string

Description

Adds a function string to a function-string class. Replication Server uses function strings to generate instructions for data servers.

Syntax

```
create function string
          [replication_definition.]function[;function_string]
          for function_class [with overwrite]
          [scan 'input_template']
          [output
          {language 'lang_output_template' |
          rpc 'execute procedure
          [@param_name=]{constant |?variable!mod?}
               [, [@param_name=]
                    {constant |?variable!mod?}]...' |
          writetext [use primary log | with log |
               no log] |
          none}]
```

Parameters

replication_definition

The name of the replication definition the function operates on. Use only for functions with replication definition scope.

Functions have either function-string-class scope or replication definition scope. Functions that direct transaction control have function-string class scope. User-defined functions, and functions that modify data, have replication definition scope.

function

The name of the function. Names for system functions must be provided as documented in Chapter 4, "Replication Server System Functions." Names for user-defined functions must match an existing user-defined function.

function_string

A function string name is required when customizing rs_get_textptr, rs_textptr_init, and rs_writetext functions, and optional for others. For rs_get_textptr, rs_textptr_init, and rs_writetext, a function string is needed for each text, unitext, or image column in the replication definition. The function string name supplied *must* be:

- The text, unitext, or image column name for the replication definition.
- Able to conform to the rules for identifiers.
- Unique in the scope of the function.

Replication Server also uses the function string name in the generation of error messages.

function_class
: The function-string class the new function string belongs to.

with overwrite
: If the function string already exists, this option drops and re-creates the function string, as though you used alter function string instead.

scan
: Precedes an input template.

input_template
: A character string, enclosed in single quote characters, that Replication Server scans to associate an rs_select or rs_select_with_lock function string with the where clause in a create subscription command. An input template string is written as a SQL select statement, with user-defined variables instead of the literal values in the subscription's where clause.

output
: Precedes an output template.

language
: Tells Replication Server to submit the output template commands to the data server using the Client/Server Interfaces language interface.

lang_output_template
: A character string, enclosed in single quote characters, that contains instructions for a data server. A language output template string may contain embedded variables, which Replication Server replaces with run-time values before it sends the string to the data server.

rpc

An output template that tells Replication Server to use the Client/Server Interfaces remote procedure call (RPC) interface. Replication Server interprets the string and constructs a remote procedure call to send to the data server.

These keywords and options appear in RPC output templates:

procedure – the name of the remote procedure to execute. It could be an Adaptive Server stored procedure, a procedure processed by an Open Server gateway RPC handler, or a registered procedure in an Open Server gateway. Refer to the *Open Server Server-Library/C Reference Manual* for information about processing RPCs in a gateway program.

@param_name – the name of an argument to the procedure, as defined by the procedure. If the *@param_name* = *value* form is used, parameters can be supplied in any order. If parameter names are omitted, parameter values must be supplied in the order defined in the remote procedure.

constant – a literal value with the datatype of the parameter it is assigned to.

?*variable*!*mod*? – *variable* is the placeholder for a run-time value. It can be a column name, the name of a system-defined variable, the name of a parameter in a user-defined function, or the name of a variable defined in an input template. The variable must refer to a value with the same datatype as the parameter it is assigned to. For a list of system-defined variables, see "System-defined variables" on page 212.

The *mod* portion of a variable name identifies the type of data the variable represents. The variable modifier is required for all variables and must be one of the following:

Table 3-23: Function string variable modifiers

Modifier	Description
new, new_raw	A reference to the new value of a column in a row you are inserting or updating
old, old_raw	A reference to the existing value of a column in a row you are updating or deleting
user, user_raw	A reference to a variable that is defined in the input template of an rs_select or rs_select_with_lock function string
sys, sys_raw	A reference to a system-defined variable
param, param_raw	A reference to a function parameter

Modifier	Description
text_status	A reference to the text_status value for text, unitext, or image data. Possible values are: • 0x000 – Text field contains NULL value, and the text pointer has not been initialized. • 0x0002 – Text pointer is initialized. • 0x0004 – Real text data will follow. • 0x0008 – No text data will follow because the text data is not replicated. • 0x0010 – The text data is not replicated but it contains NULL values.

Table 3-24: Function string variable modifiers

Modifier	Description
new, new_raw	A reference to the new value of a column in a row you are inserting or updating
old, old_raw	A reference to the existing value of a column in a row you are updating or deleting
user, user_raw	A reference to a variable that is defined in the input template of an rs_select or rs_select_with_lock function string
sys, sys_raw	A reference to a system-defined variable
param, param_raw	A reference to a function parameter
text_status	A reference to the text_status value for text, unitext, or image data. Possible values are: • 0x000 – Text field contains NULL value, and the text pointer has not been initialized. • 0x0002 – Text pointer is initialized. • 0x0004 – Real text data will follow. • 0x0008 – No text data will follow because the text data is not replicated. • 0x0010 – The text data is not replicated but it contains NULL values.

Note Function strings for user-defined functions may not use the *new* or *old* modifiers.

writetext

instructs Replication Server to use the Client-Library™ function ct_send_data to update a text, unitext, or image column value. This option applies only to the rs_writetext function.

The following options appear in writetext output templates to specify the logging behavior of the text, unitext, or image column in the replicate database:

use primary log – logs the data in the replicate database, if the logging option was specified in the primary database.

with log – logs the data in the replicate database transaction log.

no log – does not log the data in the replicate database transaction log.

none
    instructs Replication Server not to replicate a text, unitext, or image column value. This option applies only to the rs_writetext function.

Examples

Example 1 Creates a function string for the rs_begin function:

```
create function string rs_begin
 for sqlserver2_function_class
 output language
 'begin transaction'
```

Example 2 Creates a function string for the rs_commit function that contains two commands separated with a semicolon. The function string executes an Adaptive Server stored procedure that updates the rs_lastcommit system table and then commits the transaction:

```
create function string rs_commit
 for sqlserver2_function_class
 output language
 'execute sqlrs_update_lastcommit
   @origin = ?rs_origin!sys?,
   @origin_qid = ?rs_origin_qid!sys?,
   @secondary_qid = ?rs_secondary_qid!sys?;
   commit transaction'
```

Example 3 Examples 3 and 4 create a replication definition for the titles table and an rs_insert function string for the sqlserver2_function_class. The function string inserts data into the titles_rs table instead of into the titles table in the replicate database:

```
create replication definition titles_rep
 with primary at LDS.pubs2
 (title_id varchar(6), title varchar(80),
  type char(12), pub_id char(4), advance money,
  total_sales int, notes varchar(200),
  pubdate datetime, contract bit, price money)
 primary key (title_id)
 searchable columns (price)
```

Example 4 Examples 3 and 4 create a replication definition for the titles table and an rs_insert function string for the sqlserver2_function_class. The function string inserts data into the titles_rs table instead of into the titles table in the replicate database:

```
create function string titles_rep.rs_insert
 for sqlserver2_function_class
 output language
 'insert titles_rs values (?title_id!new?,
   ?title!new?, ?type!new?, ?pub_id!new?,
   ?advance!new?, ?total_sales!new?, ?notes!new?,
   ?pubdate!new?, ?contract!new?, ?price!new?)'
```

Example 5 Examples 5 and 6 create a user-defined function update_titles and a corresponding function string for the sqlserver2_function_class. The function string executes an Adaptive Server stored procedure named update_titles:

```
create function titles_rep.update_titles
 (@title_id varchar(6), title varchar(80),
  @price money)
```

Example 6 Examples 5 and 6 create a user-defined function update_titles and a corresponding function string for the sqlserver2_function_class. The function string executes an Adaptive Server stored procedure named update_titles:

```
create function string titles_rep.update_titles
 for sqlserver2_function_class
 output rpc
 'execute update_titles
   @title_id = ?title_id!param?,
   @title = ?title!param?,
   @price = ?price!param?'
```

Example 7 The rs_select function string in example 7 is used to materialize subscriptions that request rows with a specified value in the title_id column. Similar to example 8, the input templates given by the scan clauses differentiate the two function strings:

```
create function string
   titles_rep.rs_select;title_id_select
 for sqlserver2_function_class
 scan 'select * from titles
   where title_id = ?title_id!user?'
 output language
 'select * from titles
   where title_id = ?title_id!user?'
```

Example 8 The rs_select function string in example 8 is an example of an RPC function string. It is used to materialize subscriptions that request rows where the value of the price column falls within a given range:

```
create function string
```

```
  titles_rep.rs_select;price_range_select
for sqlserver2_function_class
scan 'select * from titles
  where price > ?price_min!user?
  and price < ?price_max!user?'
output rpc
  'execute titles_price_select
  ?price_min!user?, ?price_max!user?'
```

Usage

- Use create function string to add a function string to a function-string class. Function strings contain the database-specific instructions needed by Replication Server to convert a function to a command for a database.
- For an overview of functions, function strings, and function-string classes, see the *Replication Server Administration Guide Volume 2*.
- Create or alter function strings for functions with class scope at the primary site for the function-string class. See create function string class for more information about the primary site for a function-string class.
- Create or alter function strings for functions with replication definition scope, including user-defined functions, at the site where the replication definition was created. Each replication definition has its own set of function strings.
- Replication Server distributes the new function string to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.
- Some function strings are generated dynamically; they are not stored in the RSSD.

Function strings and function-string classes

- For each of the system-provided function-string classes in which a function will be used, and for each derived class that inherits from these classes, Replication Server generates a default function string for the function. This is true for both system functions and user-defined functions. (Default function strings for the rs_dumpdb and rs_dumptran functions are not provided. You only need to create them if you are using coordinated dumps.
- Customize the function string in rs_sqlserver_function_class using alter function string. Customize the function string in user-created function-string classes using create function string.

- For each user-created base function-string class in which the function will be used, and for each derived class in which you want to override the inherited function string, you must create a function string, using create function string.
- Omitting the output clause instructs Replication Server to generate a function string in the same way that it generates function strings for the rs_sqlserver_function_class or rs_default_function_class function-string classes.
- The default function string for a user-defined function is an invocation of a stored procedure where the name is the function name and the parameters are the function parameters. The stored procedure is executed as a language command, not as an RPC.

Function strings and *replicate minimal columns*

- If you have specified replicate minimal columns for a replication definition, you cannot normally create non-default function strings for the rs_update, rs_delete, rs_get_textptr, rs_textptr_init, or rs_datarow_for_writetext system functions.

  However, you can create non-default function strings for the rs_update and rs_delete functions if you use the *rs_default_fs* system variable within the function string. This variable represents the default function-string behavior. You can add additional commands to extend the function-string behavior.
- See create replication definition for more information about the replicate minimal columns option.

Input and output templates

- Depending on the function, function strings can have input and output templates. Replication Servers substitute variable values into the templates and pass the result to data servers for processing.
- Input and output templates have the following requirements:
  - They are limited to 64K. The result of substituting run-time values for embedded variables in function-string input or templates must not exceed 64K.
  - Input templates and language or RPC output templates are delimited with two single quote characters (').
  - Variable names in input templates and output templates are delimited with question marks (?).

- A variable name and its modifier are separated with an exclamation mark (!).

- When creating function strings:
  - Use two consecutive single quote characters ('') to represent one literal single quote character within or enclosing data of character or date/time datatypes, as shown for "Berkeley" in the following character string:

    ```
    'insert authors
     (city, au_id, au_lname, au_fname)
     values (''Berkeley'', ?au_id!new?,
    ?au_lname!new?,
     ?au_fname!new?)'
    ```

  - Use two consecutive question marks (??) to represent one single question mark within data of character datatypes.
  - Use two consecutive semi-colons (;;) to represent one single semi-colon within data of character datatypes.

Input templates

- Input templates are used only with the rs_select and rs_select_with_lock functions, which are used during non-bulk subscription materialization and with purge subscription dematerialization. Replication Server matches the subscription's where clause with an input template to find the function string to use.
- Input templates have the following requirements:
  - They contain only user-defined variables, whose values come from the constants in the where clause. The user-defined variables can also be referenced in the function string's output template.
  - If the *input_template* is omitted, it can match any select command. This allows you to create a default function string that is executed when no other function string in the function-string class has an *input_template* matching the select command.

Output templates

- Output templates determine the format of the command sent to a replicate data server. Most output templates can use one of two formats: language or RPC. An output template for an rs_writetext function string can use the RPC format or the additional formats writetext or none. For a description of these formats, see the *Replication Server Administration Guide Volume 2*.

- When Replication Server maps function string output templates to data server commands, it formats the variables using the format expected by Adaptive Server. It modifies datatypes for modifiers that do not end in *_raw* (the modifiers that are normally used), as follows:
  - Adds an extra single quote character to single quote characters appearing in character and date/time values to escape the special meaning of the single quote character.
  - Adds single quote characters around character and date/time values, if they are missing.
  - Adds the appropriate monetary symbol (the dollar sign in U.S. English) to values of money datatypes.
  - Adds the "0x" prefix to values of binary datatypes.
  - Adds a combination of a backslash (\) and newline character between existing instances of a backslash and newline character in character values. Adaptive Server treats a backslash followed by a newline as a continuation character, and therefore deletes the added pair of characters, leaving the original characters intact.

  Replication Server does not modify datatypes in these ways for modifiers that end in *_raw*.

  The following table summarizes how Replication Server formats each datatype for the modifiers that do not end in *_raw*:

Table 3-25: Formatting for function string variables

Datatype	Formatting of literals
bigint,int, smallint, tinyint, rs_address	Integer number
unsigned bigint, unsigned int, unsigned smallint, unsigned tinyint	Unsigned Integer number
decimal, numeric, identity	Exact decimal number
float, real	Decimal number
char, varchar	Enclosed in single quote character Adds single quote character to any instance of a single quote character Pads instances of backslash + newline characters
unichar, univarchar	Unicode
money, smallmoney	Adds the appropriate money symbol (dollar sign for U.S. English)
date, time, datetime, smalldatetime	Enclosed in single quote characters Adds single quote character to any instance of a single quote character

Datatype	Formatting of literals
binary, varbinary	Prefixed with 0x
bit	1 or 0

- Output templates have the following requirements:
  - The result of substituting run-time values for embedded variables in function-string output templates must not exceed 64K.
  - You can put several 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.

    To represent a semicolon that should not be interpreted as a command separator, use two consecutive semicolons (;;).

    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 alter connection.

System-defined variables

The following table lists system-defined variables that can be used in function-string output templates. Use the *sys* or *sys_raw* modifier for these variables:

Table 3-26: Replication Server system-defined variables

System variable	Datatype	Description
rs_default_fs	text	The default generated function-string text for the function
rs_deliver_as_name	varchar(200)	For execution of a replicated function, name of the procedure to be invoked at the destination
rs_destination_db	varchar(30)	Name of the database where a transaction was sent
rs_destination_ds	varchar(30)	Name of the data server where a transaction was sent
rs_destination_ldb	varchar(30)	Name of the logical database where a transaction was sent
rs_destination_lds	varchar(30)	Name of the logical data server where a transaction was sent
rs_destination_ptype	char(1)	Physical connection type ("A" for active or "S" for standby) for the database where a transaction was sent
rs_destination_user	varchar(30)	User who will execute the transaction at the destination
rs_dump_dbname	varchar(30)	Name of the database where a database or transaction dump originated
rs_dump_label	varchar(30)	Label information for a database or transaction dump. For Adaptive Server, this variable holds a datetime value that is the time the dump originated.

System variable	Datatype	Description
rs_dump_timestamp	varbinary(16)	Timestamp of a database or transaction dump
rs_lorigin	int(4)	ID of the originating logical database for a transaction
rs_isolation_level	varchar(30)	Transaction isolation level of a database connection.
rs_origin	int(4)	ID of the originating database for a transaction
rs_origin_begin_time	datetime	The time that a command was applied at the origin **Note** If you execute select getdate() while ASE is still processing user database recovery, the returned value of select getdate() may be different from the value of rs_origin_begin_time.
rs_origin_commit_time	datetime	The time that a transaction was committed at the origin **Note** If you execute select getdate() while ASE is still processing user database recovery, the returned value of select getdate() may be different from the value of rs_origin_begin_time.
rs_origin_db	varchar(30)	Name of the origin database
rs_origin_ds	varchar(30)	Name of the origin data server
rs_origin_ldb	varchar(30)	Name of the logical database for a warm standby application
rs_origin_lds	varchar(30)	Name of the logical data server for a warm standby application
rs_origin_qid	varbinary(36)	Origin queue ID of the first command in a transaction
rs_origin_user	varchar(30)	User who executed the transaction at the origin
rs_origin_xact_id	binary(120)	The system-assigned unique ID of a transaction
rs_origin_xact_name	varchar(30)	User-assigned name of the transaction at origin
rs_secondary_qid	varbinary(36)	Queue ID of a transaction in a subscription materialization or dematerialization queue
rs_last_text_chunk	int(4)	If the value is 0, this is not the last chunk of text data. If the value is 1, this is the last chunk of text data.
rs_writetext_log	int(4)	If the value is 0, rs_writetext has not finished logging text, unitext, and image data at the primary database transaction log. If the value is 1, rs_writetext has finished logging text, unitext, and image data at the primary database transaction log.

rs_origin_commit_time system variable

If you are not using parallel DSI to process large transactions before their commit has been read from the DSI queue, the value of the *rs_origin_commit_time* system variable contains the time when the last transaction in the transaction group committed at the primary site.

If you are using parallel DSI to process large transactions before their commit has been read from the DSI queue, when the DSI threads start processing one of these transactions, the value of the *rs_origin_commit_time* system variable is set to the value of the *rs_origin_begin_time* system variable. When the commit statement for the transaction is read, the value of *rs_origin_commit_time* is set to the actual commit time. Therefore, when the configuration parameter dsi_num_large_xact_threads is set to a value greater than zero, the value for *rs_origin_commit_time* is not reliable for any system function other than rs_commit.

System variables and NULL values

- The following system variables may have NULL values:
    - *rs_origin_ds*
    - *rs_origin_db*
    - *rs_origin_user*
    - *rs_origin_xact_name*
    - *rs_destination_db*
    - *rs_destination_user*
    - *rs_dump_dbname*
    - *rs_dump_label*

  When a system variable has no value, Replication Server maps the word "NULL" into function-string templates. This may cause syntax errors in some generated statements. For example, the following command would be generated if *rs_origin_xact_name* has a null value:

  ```
  begin transaction NULL
  ```

  To prevent this error, create a function string with an output template like the following:

  ```
  'begin transaction t_?rs_origin_xact_name!sys_raw?'
  ```

  If the *rs_origin_xact_name* system variable is null, the transaction name will be "t_NULL".

Replacing function strings

- To replace a function string, use alter function string or create function string with overwrite. Either approach executes drop function string and create function string in a single transaction, preventing errors that could result from temporarily missing function strings.

Permissions	create function string requires “create object” permission.
See also	alter function string,configure connection, create connection, create function string class, create subscription, define subscription, drop function string, alter function string

create function string class

Description

Creates a function-string class.

Syntax

create function string class *function_class*
    [set parent to *parent_class*]

Parameters

function_class
The name of the function-string class to create. It must conform to the rules for identifiers. Function-string class names have a global name space, so they must be unique in the replication system.

set parent to
Designates a parent class for a new derived class.

parent_class
The name of an existing function-string class you designate as the parent class for a new derived class. rs_sqlserver_function_class cannot be used as a parent class.

Examples

Example 1 Creates a derived function-string class named sqlserver_derived_class that will inherit function strings from the system-provided class rs_default_function_class:

```
create function string class
 sqlserver_derived_class
 set parent to rs_default_function_class
```

Example 2 Creates a function-string class named sqlserver2_function_class. This class will be a base class, and will not inherit function strings. You can, however, specify this class as a parent class for a derived class:

```
create function string class sqlserver2_function_class
```

Usage

- Use create function string class to create a function-string class. function-string classes group function strings for a database. The function-string class, with its member function strings, is associated with a database. This association is made with the create connection or alter connection command.
- The Replication Server to which create function string class is sent becomes the primary Replication Server for the newly-created function-string class.
- Create a new derived class using the set parent to clause to specify a parent class from which the new class is to inherit function strings. Omit this clause to create a new base class, which does not inherit function strings from a parent class.

- For an overview of function-string classes, function strings, and functions, see the *Replication Server Administration Guide Volume 2*.
- Before you execute this command, make sure that the name for the new function-string class is unique in the replication system. Replication Server does not detect all name conflicts.
- Replication Server distributes the new function-string class to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.
- To modify function strings in the class rs_sqlserver_function_class, you must first select a Replication Server to be the primary site for the class. Then execute create function string class for rs_sqlserver_function_class at that site.
- The Replication Server that serves as the primary site for any function-string class must have routes to all other Replication Servers where the class will be used.
- The primary site for a derived class is the same as the primary site of its parent class. You must create a derived class at the primary site of its parent class. However, if the parent class is a system-provided class, rs_default_function_class or rs_db2_function_class, the primary site of the derived class is the Replication Server where you created the derived class.

System-provided function-string classes

- Replication Server provides three function-string classes that you can use:
  - rs_sqlserver_function_class – default generated Adaptive Server function strings are provided for this class. The default function strings in rs_sqlserver_function_class are identical to those in rs_default_function_class. This class is assigned by default to Adaptive Server databases you add to the replication system using rs_init. You can customize function strings for this class. rs_sqlserver_function_class cannot be used as a parent class or a derived class.
  - rs_default_function_class – default generated Adaptive Server function strings are provided for this class. The default function strings in rs_sqlserver_function_class are identical to those in rs_default_function_class. You cannot customize function strings for this class. This class can be used as a parent class but cannot become a derived class.

- rs_db2_function_class – default generated DB2-specific function strings are provided for this class. Although this class is a derived class of rs_default_function_class, with customizations for DB2, you cannot customize function strings for this class. rs_db2_function_class can be used as a parent class but cannot be made a derived class.

Benefits of function-string inheritance

- Using derived classes that inherit from system-provided classes rs_default_function_class or rs_db2_function_class, either directly or indirectly, allows you to customize only the function strings you want to customize and inherit all others, even for new table or function replication definitions.

  If you use classes that do not inherit from system-provided classes, you must create all function strings yourself, either in parent or derived classes, and add new function strings whenever you create a new table or function replication definition.

- After an upgrade to a future release of Replication Server, derived classes that inherit from the system-provided classes rs_default_function_class or rs_db2_function_class, either directly or indirectly, will inherit function-string definitions for any new system functions.

Adding function strings to a function-string class

- After you create a function-string class that does not inherit function strings from a parent class, add function strings for the system functions that have function-string-class scope. Then add function strings for system functions and user-defined functions that have replication definition scope and will be replicated to databases that use the new function-string class.
- To create or customize function strings in a function-string class, use create function string. You cannot create function strings in the classes rs_default_function_class or rs_db2_function_class.

Permissions

create function string class requires "sa" permission.

See also

alter connection, alter function string class, create connection, create function, create function string class, move primary

create logical connection

Description

Creates a logical connection. Replication Server uses logical connections to manage warm standby applications.

Syntax

create logical connection to *data_server*.*database*
[set *logical_database_param* [to] '*value*'
[set *logical_database_param* [to] '*value*']...]

Parameters

data_server
The name of a data server. The data server does not have to be a real data server.

database
The name of a database. The database does not have to be a real database.

logical_database_param
The name of the configuration parameter that affects logical connections. Table 3-15 describes the parameters that you can set with create logical connection.

Examples

Example 1 Creates a logical connection called LDS.logical_pubs2:

```
create logical connection to LDS.logical_pubs2
```

Example 2 Creates a logical connection for an existing connection. For example, you would enter this if the database TOKYO_DS.pubs2 already exists and will serve as the active database in the warm standby application:

```
create logical connection to TOKYO_DS.pubs2
```

Usage

- create logical connection creates a logical connection to be used with warm standby applications. See the *Replication Server Administration Guide Volume 2* for information about setting up and managing warm standby applications.
- The logical connection is for a symbolic *data_server*.*database* specification. The data server and database do not have to be real; Replication Server maps them to the current active database.
- If you are creating a logical connection for an existing connection, *data_server*.*database must* refer to the data server and database names of the existing connection. Otherwise, it is recommended that the logical name be different from the active and standby database names.
- Replication definitions and subscriptions use the logical connection name.
- After you create the logical connection, use rs_init to add the physical active and standby databases for the logical connection.

Permissions	create logical connection requires “sa” permission.
See also	alter logical connection, configure connection, configure logical connection, drop connection, drop logical connection, switch active

create partition

Description

Makes a partition available to Replication Server. A partition can be a disk partition or an operating system file.

Syntax

create partition *logical_name*
on '*physical_name*' with size *size*
[starting at *vstart*]

Parameters

logical_name
A name for the partition. The name must conform to the rules for identifiers. The name is also used in the drop partition and alter partition commands.

physical_name
The full specification of the partition. This name must be enclosed in single quotation marks.

size
The size, in megabytes, of the partition. Maximum size possible is 1TB.

starting at *vstart*
Specifies the number of megabytes (*vstart*) to offset from the beginning of the partition.

Examples

Example 1 Adds a 20MB partition named P1 on the device named */dev/rsd0a*:

```
create partition P1 on '/dev/rsd0a' with size 20
```

Example 2 Adds a 20MB partition named P1 on the device named */dev/rsd0a*. Since an offset of 1MB is specified, however, the total usable partition space available to Replication Server is 19MB:

```
create partition P1 on '/dev/rsd0a' with size 20
 starting at 1
```

Usage

- Replication Server uses partitions for stable message queues. The message queues hold data until it is sent to its destination.
- Increasing the available disk space in partitions allows Replication Server to support more routes and database connections and to continue to queue messages during longer failures.
- The maximum size for a partition is 1TB, which is approximately 1,000,000MB.
- Disk partitions must not be mounted for use by the operating system and should not be used for any other purpose, such as for swap space or an Adaptive Server disk device.

- Allocate the entire partition to Replication Server. If you allocate part of a partition for Replication Server, you cannot use the remainder for any other purpose. If you use the starting at *vstart* clause, the partition space available to Replication Server is what is left after you subtract the offset size from the total partition size.
- The starting at *vstart* clause makes space available at the beginning of the partition for disk mirroring information.
- You can use operating system files for partitions. However, the operating system buffers file I/O, so you may not be able to recover stable queues completely following a failure. Therefore, you should only use files for partitions in a test environment—unless your operating system does not support physical disk partitions.
- If you use an operating system file, you must create it before executing create partition. On UNIX platforms, you can create the file with the touch command. The file can be zero bytes in length; the create partition command extends the file to the specified *size*.
- The "sybase" user should own the disk partition or operating system file and must have read and write permissions on it. Users other than "sybase" should not have write or read permission on the partition.

Permissions

create partition requires "sa" permission.

See also

admin disk_space, drop partition, alter partition

create publication

Description

Creates a publication for tables or stored procedures that are to be replicated as a group to one or more subscribing replicate databases.

Syntax

create publication *pub_name*
    with primary at *data_server*.*database*

Parameters

pub_name
A name for the publication. It must conform to the rules for identifiers and be unique for the specified primary data server and database.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

Examples

Creates a publication called pubs2_pub that you can use to replicate data for multiple tables and stored procedures in the pubs2 database.

```
create publication pubs2_pub
 with primary at TOKYO_DS.pubs2
```

Usage

- Use create publication to create a publication, an object that makes it easy to set up replication for multiple tables or stored procedures in a database. You create a publication, add articles, which specify replication definitions, and then create a single subscription for the publication.
- Execute create publication at the Replication Server that manages the database where the primary data is stored.
- For more information about working with replication definitions, articles, and publications, see the *Replication Server Administration Guide Volume 1*.

  For more information about subscribing to publications, refer to Chapter 10, "Managing Subscriptions," in the same book.
- Replication Server distributes information about a new publication to a replicate site only when you create or refresh a subscription for the publication.

Requirements for using *create publication*

- Before executing create publication, make sure that:
  - The publication name you enter is unique for the primary data server and database.
  - A connection exists from the Replication Server to the database where the primary tables or stored procedures are stored.

- A Replication Server of release 11.0.x does not receive information about publications or articles and cannot subscribe to them.

Preparing publications for subscription

- After you create a publication, you use create article to create articles and assign them to the publication. An article specifies a table replication definition or function replication definition and includes optional where clauses according to the needs of the subscribing replicate site. See create article for more information.
- Because a replicate table cannot subscribe to two or more replication definitions for the same primary object, a publication cannot contain two or more articles for different replication definitions for the same primary table and the same replicate table.
- When all of the articles have been assigned, you must validate the publication using validate publication before a replicate site can subscribe to it. Validating a publication verifies that the publication contains at least one article and marks the publication ready for subscription. See validate publication for more information.
- To check publication status, use check publication. This command displays the number of articles the publication contains and indicates if the publication is valid. See check publication for more information.

Subscribing to publications

- When a publication is valid, you can create a subscription for the publication in order to begin replication to a replicate database. All forms of subscription materialization are supported. See create subscription or define subscription for more information.
- When you create a publication subscription, Replication Server creates a separate underlying subscription for each article that the publication contains. Each article subscription uses the name of the parent publication subscription.
- A subscription to a publication cannot include a where clause. Instead, you can customize replication to replicate sites by including one or more where clauses in each article the publication contains.

Articles for table replication definitions

- If a publication contains articles for table replication definitions only, you can use create subscription to subscribe to the publication using atomic or non-atomic materialization. See create subscription for more information.
- You can also use bulk materialization for the publication subscription:

  - When data already exists at the replicate database, use create subscription with the without materialization clause.
  - When you must manually transfer subscription data, use define subscription and the other bulk materialization commands. See define subscription for more information.

Articles for function replication definitions

- If a publication contains articles for function replication definitions only, use bulk materialization for the publication subscription:
  - When data already exists at the replicate database, use create subscription with the without materialization clause. See create subscription for more information.
  - When you must manually transfer subscription data, use define subscription, activate subscription, and validate subscription to subscribe to the publication using bulk materialization. See define subscription for more information.

Articles for both table and function replication definitions

- If a publication contains articles for both table replication definitions and function replication definitions, you can use the same subscription command even though each type of replication definition requires a different materialization method.

  In order to create the subscription, first transfer data to the replicate database for component subscriptions that require bulk materialization, such as those for function replication definitions. Then use create subscription to subscribe to the publication:
  - Subscriptions for articles for table replication definitions are materialized using atomic or non-atomic materialization—unless you use the without materialization clause.
  - Subscriptions for articles for function replication definitions are materialized without materialization.

    In cases where the stored procedure for a function replication definition operates on a table for which there is also a table replication definition, no separate data transfer is necessary.

Refreshing publication subscriptions

- If you add a new article to an existing publication, or drop an article from the publication, the publication is invalidated. Although replication for existing articles continues unaffected, in order to begin replication for any new articles or create new publication subscriptions you must:

    - Validate the publication when you have completed making changes to the publication, then
    - Refresh the publication subscription.
- In order to refresh a publication subscription for atomic or non-atomic materialization:
    - Re-create the subscription using create subscription. See create subscription for more information.
- In order to refresh a publication subscription for bulk materialization:
    - When data already exists at the replicate database, use create subscription with the without materialization clause.
    - Re-create the subscription using define subscription, activate subscription, and validate subscription and transfer subscription data manually as necessary. See define subscription for more information.

Dropping subscriptions, articles, and publications

- You can drop a subscription to a publication and, optionally, purge the subscription data for the component subscriptions to articles for table replication definitions. See drop subscription.
- If there is no subscription, you can drop an article that a publication contains and, optionally, drop the associated replication definition if it is not used elsewhere. After you drop an article, the publication is invalid. See drop article.
- You can drop a publication if there are no subscriptions for the publication. When you drop a publication, its articles are also dropped. Optionally, you can also drop all of the replication definitions for the publication's articles, if they are not used elsewhere. See drop publication for more information.

Publications in warm standby applications

- In a warm standby application, replication definitions used in replicating to the standby database may also be specified by articles included in publications.

Permissions

create publication requires "create object" permission.

See also

check publication, create article, create function replication definition, create replication definition, create subscription, define subscription, drop article, drop publication, drop subscription, validate publication.

create replication definition

Description

Creates a replication definition for a table that is to be replicated.

Syntax

```
create replication definition replication_definition
with primary at data_server.database
[with all tables named [table_owner.] 'table_name' |
[with primary table named [table_owner.]'table_name']
with replicate table named
        [table_owner.]'table_name']]
(column_name [as replicate_column_name]
        [datatype [null | not null]
        [map to published_datatype]]
[, column_name [as replicate_column_name]
        [datatype [null | not null] computed]
        [map to published_datatype ]]...)
primary key (column_name [, column_name]...)
[searchable columns (column_name
        [, column_name]...)]
[send standby [{all | replication definition}
        columns]]
[replicate {minimal | all} columns]
[replicate_if_changed
        (column_name [, column_name]...)]
[always_replicate (column_name [, column_name]...)]
```

Parameters

replication_definition
The replication definition, which must conform to the rules for identifiers. The replication definition name is assumed to be the name of both the primary and replicate tables, unless you specify the table names.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

with all tables named
Specifies the table name at both the primary and replicate databases. *table_name* is a character string of up to 200 characters. *table_owner* is optional, and represents the table owner. Data server operations may fail if the actual table owners do not correspond to what you specify in the replication definition.

with primary table named

Specifies the table name at the primary database. *table_name* is a character string of up to 200 characters. *table_owner* is optional and represents the table owner. Data server operations may fail if the actual table owners do not correspond to what you specify in the replication definition.

If you specify the primary table name but do not also specify the replicate table name, the replication definition name is assumed to be the name of the replicate table.

with replicate table named

Specifies the name of the table at the replicate database. *table_name* is a character string of up to 200 characters. *table_owner* is optional and represents the table owner. Data server operations may fail if the actual table owners do not correspond to what you specify in the replication definition.

If you specify the replicate table name but do not also specify the primary table name, the replication definition name is assumed to be the name of the primary table.

column_name

A column name from the primary table. You cannot use a column name more than once in each clause.

Each column and datatypes must be enclosed in parentheses ( ).

as *replicate_column_name*

Specifies a column name in a replicate table into which data from the primary column will be copied. Use this clause when the source and destination columns have different names.

datatype

The datatype of the column in the primary table. See “Datatypes” on page 21 for a list of the datatypes and syntax.

Use as *declared_datatype* if you are specifying a column-level datatype translation. A declared datatype must be a native Replication Server datatype or a datatype definition for the primary datatype.

For different replication definitions created against the same table, the column datatypes must be the same, however the published datatypes may be different. See the *Replication Server Administration Guide Volume 1* for more information.

Specifying the datatype is optional if a replication definition created against the same table already has this column.

null or not null

Applies only to text, unitext, image, or rawobject columns. Specifies whether a null value is allowed in the replicate table. The default is not null, meaning that the replicate table does not accept null values.

The null status for each text, unitext, image, and rawobject column must match for all replication definitions for the same primary table, and must match the settings in the actual tables. Specifying the null status is optional if an existing replication definition of the same primary table has text, unitext, image, and rawobject columns.

You cannot change this setting for a column once it is included in a replication definition for the table. To change the value, you must drop and re-create all replication definitions that include the column.

map to *published_datatype*

Specifies the datatype of a column after a column-level datatype translation, but before any class-level translation and before presentation to the replicate database.

primary key *column_name*

Specifies the columns that form the primary key for the table. You cannot use a column name more than once in each clause.

You cannot include text, unitext, image, rawobject, rawobject in row, or rs_address columns as part of the primary key.

searchable columns *column_name*

Specifies the columns that can be used in where clauses of create subscription, define subscription, or create article. You cannot use a column name more than once in each clause.

You cannot specify text, unitext, image, rawobject, rawobject in row or encrypted columns as searchable columns.

send standby

Specifies how to use the replication definition in replicating into a standby database in a warm standby application. See "Replication definitions and warm standby applications" on page 237 for details on using this clause and its options.

replicate minimal columns or replicate all columns

Sends all replication definition columns for every transaction or only those needed to perform update or delete operations at replicate databases. The default is to replicate all columns.

replicate_if_changed

Replicate text, unitext, image, or rawobject columns only when their column data changes.

always_replicate

Always replicate text, unitext, image, and rawobject columns.

Examples

Example 1 Creates a replication definition named authors_rep for the authors table. The primary copy of the authors table is in the pubs2 database in the LDS data server. All copies of the table are also named authors. Only the minimum number of columns will be replicated for delete and update operations:

```
create replication definition authors_rep
  with primary at LDS.pubs2
  with all tables named 'authors'
    (au_id varchar(11), au_lname varchar(40),
     au_fname varchar(20), phone char(12),
     address varchar(12), city varchar(20),
     state char(2), country varchar(12), postalcode
     char(10))
  primary key (au_id)
  searchable columns (au_id, au_lname)
  replicate minimal columns
```

Example 2 Creates a replication definition called blurbs_rep for the blurbs table owned by “emily” in the pubs2 database. Data in the copy column, which uses the text datatype and accepts null values, will be replicated when the column data changes:

```
create replication definition blurbs_rep
  with primary at TOKYO_DS.pubs2
  with all tables named emily.'blurbs'
    (au_id char(12), copy text null)
  primary key (au_id)
  replicate_if_changed (copy)
```

Example 3 Where at least one replication definition already exists for the primary table publishers in the pubs2 database, this command creates an additional replication definition called pubs_copy_rep. This replication definition can be subscribed to by replicate tables that are named pubs_copy and for which “joe” is the owner. Subscriptions may fail for replicate tables that are also named pubs_copy but for which “joe” is not the owner:

```
create replication definition pubs_copy_rep
  with primary at TOKYO_DS.pubs2
  with primary table named 'publishers'
```

```
      with replicate table named joe.'pubs_copy'
        (pub_id, pub_name as pub_name_set)
      primary key (pub_id)
```

Data for the pub_name column in the primary table will replicate into the pub_name_set column in the replicate table, which must share the same datatype. You do not need to specify the datatype for a column in an existing replication definition. In this example, the city and state columns from the primary table are not required for the replicate table pubs_copy, and are thus excluded from this replication definition.

Example 4 Creates a replication definition that replicates all modified columns of the authors table to the standby database. This definition also replicates to the MSA, however, only the modified values of au_id and au_lname columns are replicated. au_id is the key used to update and delete from the authors table:

```
create replication definition authors_rep
  with primary at LDS.pubs2
  with all tables named 'authors'
     (au_id varchar(11), au_lname varchar(40))
  primary key (au_id)
  send standby
  replicate minimal columns
```

Usage

- Execute this command at the Replication Server that manages the database where the primary version of the table is stored.
- Use rs_helprep to determine which replication definitions are available to Replication Server version 12.0 and earlier. For more information, see rs_helprep on page 483.
- For an overview of defining and maintaining replicated tables, and for information about working with replication definitions, articles, and publications, see the *Replication Server Administration Guide Volume 1.*
- Before executing the create replication definition command, be sure that:
    - The replication definition name you enter is unique among all replication definitions (table or function) in the replication system. Replication Server cannot always enforce this requirement when you enter create replication definition.
    - A connection exists from the Replication Server to the database where the primary table is stored. See create connection for more information. You can also create database connections using rs_init. See the Replication Server installation and configuration guides for your platform.

    - If you use more than one version of Replication Server (for example, version 12.0 and version 11.0.x) and you create multiple replication definitions for the same primary table, review any mixed-version issues for your replication system (for example, if column names are different for the same table in both versions). See "Creating multiple replication definitions" on page 232 for details.

- Replication Server distributes the new replication definition to qualifying sites through the replication system. The changes do not appear immediately at all sites because of normal replication system lag time.

Replication status

- The replication status for text, unitext, image, and rawobject columns must be the same in the Adaptive Server database and in the replication definition.
- Use alter replication definition to change replication status.
- The replication status must be consistent for all of the replication definitions created against the same primary table.
    - If you change the replication status using alter replication definition, the replication status for other replication definitions against the same primary table also changes.
    - You do not have to specify replication status if the column is already listed in another replication definition for the same primary table.

Creating multiple replication definitions

- You can create multiple replication definitions for the same primary table and customize each one so it can be subscribed to by a replicate table whose characteristics are different from those of the primary table and other replicate tables.

  In addition to describing the primary table, each replication definition can specify, for example, a smaller number of columns, different column names, or a different table name for a replicate table. Replicate tables that match the specified characteristics can subscribe to the replication definition. You can also use multiple replication definitions even when replicate and primary tables match.

  This feature also allows you to create one replication definition for normal replication and another one for standby if the database requirements are different. See the *Replication Server Administration Guide Volume 1* for details.

- A replicate table can subscribe to only one replication definition per primary table, although it can subscribe to the same replication definition more than once.
- Different replication definitions created for the same primary table must use the same column datatype and the same null status for text, unitext, and image columns.
- If a replication definition specifies different primary and replicate table names, specifies different primary and replicate column names, or includes table owner names, only Replication Servers version 11.5 or later can subscribe to it. Such a replication definition is incompatible with Replication Servers version 11.0.*x* or earlier.
- The first replication definition created for a table is marked and propagated to a Replication Server of a version earlier than 11.5 if it is compatible; that is, if it has the same primary and replicate table and column names, and does not include the table owner name.
- If a table is replicated to standby or MSA connection using internal replication definition and dynamic SQL is enabled for the connection, any new replication definition for the table should define the column order consistent with the column order in the primary database. Otherwise, it may invalidate the existing prepared statements and may require the standby or MSA connection to be restarted.

Functions and function strings

- Replication Server creates rs_insert, rs_delete, rs_update, rs_truncate, rs_select, and rs_select_with_lock functions for the replication definition. If the replication definition contains text, unitext, image, or rawobject data, Replication Server also creates rs_datarow_for_writetext, rs_get_textptr, rs_textptr_init, and rs_writetext functions.
- Replication Server generates default function strings for these functions for the system-provided function-string classes and for derived classes that inherit from these classes. Some function strings may be generated dynamically, so they never exist in the RSSD. For other function-string classes, you must create all the function strings.
- For each function-string class, each replication definition for the same table has its own set of function strings for the system functions.
- When you create, drop, or alter a user-defined function, it is created, dropped, or altered for all the replication definitions for the same primary table.

- Although different replication definitions for the same primary table share the same user-defined functions, each user-defined function has its own function string. You create user-defined functions using create function when you replicate stored procedures using the method associated with table replication definitions.

Specifying columns and datatypes

- When you specify the columns and datatypes you want to replicate, observe these guidelines:
  - Columns that have user-defined datatypes must be defined in the replication definition with the underlying base datatypes.
  - The replication status (that is, replicate_if_changed, always_replicate) of a text, unitext, image, or rawobject column must be the same for all replication definitions on the primary table. If you change a text, unitext, image, or rawobject column's replication status using alter replication definition, the replication status for that column also changes for other replication definitions for the same primary table.

    You do not have to specify the replication status of a text, unitext, image, or rawobject column that is part of a replication definition for the same table.
  - Omit length and precision from numeric datatype declarations. Replication Server processes numeric datatype values without affecting precision.

    Note If you use the map to option to translate a larger varchar to a varchar with a smaller number of characters per column, make sure that any data you replicate does not exceed the character length of the column you replicate to.

    For instance, you can map a varchar(100) to a varchar(25) column, as long as the item you replicate does not exceed the limit of varchar(25). If it does, an error message appears.
  - Declare columns with the Adaptive Server timestamp datatype as binary(8) in replication definitions. See "Datatypes" on page 21 for more information about mapping Adaptive Server datatypes to supported datatypes.

- If a replication definition column list contains an IDENTITY column and the replicate table is in Adaptive Server, the maintenance user must be the owner of the table (or must be "dbo" or aliased to "dbo") at the replicate database in order to use the Transact-SQL identity_insert option.

  A primary table (with one or multiple replication definitions) can contain only one IDENTITY column. However, you may use the map to option to publish multiple columns as the identity datatype with one or multiple replication definitions.

- The rs_address datatype allows a unique subscription resolution technique. Bitmaps of the rs_address datatype (based on the underlying int datatype) are compared with a bitmask in a subscription's where clause to determine whether a row should be replicated. To use this subscription resolution method, you must first create tables that use columns of the int datatype. In creating a replication definition, include these columns in the column list, but declare the datatype to be rs_address instead of int.

  See create subscription for more information. Also, see the *Replication Server Administration Guide Volume 1* for more information about using the rs_address datatype.

Specifying columns and datatypes for column-level translations

- You cannot use text, unitext, image, or rawobject datatypes as a base datatype or a datatype definition or as a source or target of either a column-level or class-level translation.
- *declared_datatype* depends on the datatype of the value delivered to Replication Server:
  - If the Replication Agent delivers a native Replication Server datatype, *declared_datatype* is the native datatype.
  - If the Replication Agent delivers any other datatype, *declared_datatype* must be the datatype definition for the original datatype in the primary database.
- *published_datatype* is the datatype of the value after a column-level translation, but before any class-level translation. *published_datatype* must be a Replication Server native datatype or a datatype definition for the datatype in the target database.
- Columns declared in multiple replication definitions must use the same *declared_datatype* in each replication definition.The *published_datatype* can differ.

Using the *replicate minimal columns* option

- Using the replicate minimal columns option can improve DSI performance, reduce message overhead, and reduce queue size. It can also help to avoid application problems caused by triggers that are set for columns that are not actually changed.

  For details on how this option works, see the *Replication Server Administration Guide Volume 2*.

- The following requirements apply to replicating minimal columns:

  - Normally, replicate minimal columns can be used only with replication definitions that use the default function strings for the rs_update and rs_delete functions. If you specify replicate minimal columns, you can create non-default rs_update and rs_delete function strings for the replication definition using the *rs_default_fs* system variable within the function string. See create function string for details.

  - You cannot use autocorrection with the replicate minimal columns option. If you specify set autocorrection on before you set replicate minimal columns, an informational message is logged for each delete or update operation. If you first specify replicate minimal columns, you cannot specify set autocorrection on for the replication definition.

  - If you have specified replicate minimal columns for a replication definition, you cannot create a subscription for it using non-atomic materialization (create subscription command, without holdlock option), or use the bulk materialization option that simulates non-atomic materialization. See the *Replication Server Administration Guide Volume 2* for more information.

Replicating *text*, *unitext*, *image*, or *rawobject* datatypes

- The primary key of the replication definition must include the column or columns that uniquely identify a single row in the table.

- The always_replicate and replicate_if_changed clauses let you specify the replication status for text, unitext, image, and rawobject columns. You can also set this status in the Adaptive Server system procedures sp_setreptable and/or sp_setrepcol, or sp_reptostandby. The replication status must be the same in the Adaptive Server system procedures and in the replication definitions of a primary table. If there are inconsistencies, the RepAgent can shut down. See the *Replication Server Administration Guide Volume 1* for information on setting status and resolving inconsistencies if they occur. See "Replication definitions and warm standby applications" on page 237 for information about replicating text, unitext, image, and rawobject data into warm standby applications.
- You must specify the replication definition's replication status as always_replicate when you mark a table with sp_setreptable only, because the sp_setreptable default replication status is always_replicate. You can change a table's replication status to replicate_if_changed by changing the table's replication definition replication status to replicate_if_changed and marking every column in the table with the sp_setrepcol replication status set to replicate_if_changed.
- The following requirements apply to replicating text, unitext, image, or rawobject datatypes:
  - If a text, unitext, image, or rawobject column appears in the replicate_if_changed column list, attempting to enable autocorrection for the replication definition will cause an error. Autocorrection requires that all text, unitext, image, and rawobject columns appear in the always_replicate list for the replication definition.
  - If a text, unitext, image, or rawobject column with replicate_if_changed status was not changed in an update operation at the primary table and the update causes the row to migrate into a subscription, the inserted row at the replicate table will be missing the text, unitext, image, or rawobject data. Replication Server displays a warning message in the error log when the row migrates into the subscription and the text, unitext, image, or rawobject data is missing. In this case, run rs_subcmp to reconcile the data in the replicate and primary tables.

Replication definitions and warm standby applications

- Replication Server does not require replication definitions to maintain a standby database in a warm standby application. Using replication definitions may improve performance in replicating into the standby database. You can create a replication definition just for this purpose for each table in the logical database.

- Use send standby with any option to use the replication definition to replicate transactions for the table to the standby database. The replication definition's primary key columns and replicate minimal columns setting are used to replicate into the standby database. The options for this method include:
  - Use send standby or send standby all columns to replicate all columns in the table to the standby database.
  - Use send standby replication definition columns to replicate only the replication definition's columns to the standby database.
- Use send standby off in alter replication definition to indicate that you don't want any single replication definition for this table to be used in replicating into the standby database.

  When none of a primary table's replication definitions are marked as used by the standby, all columns are replicated into the standby database, the union of all primary keys for all replication definitions for the table is used for the primary key, and minimal columns are replicated. The replicate_minimal_columns setting for the logical connection determines whether to send minimal columns or all columns for update and delete. See alter logical connection and alter replication definition for details.
- See the *Replication Server Administration Guide Volume 2* for more information about the performance optimizations gained by using replication definitions for replicating into the standby database.
- In a primary table with multiple replication definitions, if a replication definition is already marked as used by the standby, another replication definition created or altered with send standby unmarks the first one.
- You must specify the replication definition's replication status as replicate_if_changed when you mark a database with sp_reptostandby only, because the sp_reptostandby default replication status is replicate_if_changed. You cannot change the replication status of text, unitext, image, and rawobject columns when the database is marked with sp_reptostandby only.
- When you mark a database with sp_reptostandby and a table in that database with sp_setreptable, you must specify the replication status for the replication definition as always_replicate—because the default replication status is always_replicate. You can change a table's replication status to replicate_if_changed by changing the table's replication definition replication status to replicate_if_changed and marking every column in the table with the sp_setrepcol replication status set to replicate_if_changed.

Altering replication definitions

- Use alter replication definition to add more columns or more searchable columns and to make other changes to the settings for an existing replications definition. See alter replication definition for details.
- If you need to remove or rename primary columns in an existing replication definition, you must drop all subscriptions to the replication definition, drop the replication definition and re-create it, and re-create the subscriptions.

Replicating stored procedures

- To enable replication of stored procedures, use create function replication definition. For an overview of replicating stored procedures, see the *Replication Server Administration Guide Volume 1*.

Replicating computed columns

- create replication definition supports the replication of materialzied computed columns. Materialized computed columns need to be defined using its base datatype in the replication definition.
- Materialized computed column is a computed column whose value is stored in the table page same as regular columns. It is re-evaluated upon each insert or update on its base column. It is not re-evaluated in a query.
- There is another type of computed column called virtual or non-materialized computed column. The value of this computed column is not stored in the table or an index. It is only evaluated when it is referenced in a query and no action is taken upon insert or update operation.

  Replication of virtual computed columns is not supported and this should not be included in the replication definition.

For more information on replicating computed columns, see *Replication Server Administration Guide Volume 1.*

Permissions

create replication definition requires “create object” permission.

See also

alter function string, alter replication definition, configure logical connection, create connection, create function replication definition, create function string, create subscription, drop replication definition, set autocorrection, sp_setrepcol, sp_setreptable

create route

Description

Designates the route to use for a connection from the current Replication Server to a remote Replication Server.

Syntax

```
create route to dest_replication_server {
 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']... ]}
```

Parameters

dest_replication_server
The destination Replication Server.

thru_replication_server
The intermediate Replication Server through which to pass messages for the destination Replication Server. Specify this when creating an indirect route.

user
The Replication Server login name to use to log in to the destination Replication Server. This is the login name used by the RSI user thread. If no user name is entered, Replication Server uses the principal user name entered with the -S flag when Replication Server was started.

passwd
The password to use with the login name. If no password is entered, Replication uses a null value.

route_param
a parameter that affects routes. See Table 3-16 on page 147 for a list of parameters and values.

value
a character string containing a value for a parameter.

security_param
Specifies the name of a security parameter. Refer to Table 3-27 on page 240 for a list and description of security parameters that can be set with create route.

Table 3-27: Parameters affecting network-based security

security_param	value
msg_confidentiality	Indicates whether Replication Server sends and receives encrypted data. If set to “required,” outgoing data is encrypted. If set to “not required,” Replication Server accepts incoming data that is encrypted or not encrypted. Default: not_required

security_param	value
msg_integrity	Indicates whether data is checked for tampering. Default: not_required
msg_origin_check	Indicates whether the source of data should be verified. Default: not_required
msg_replay_detection	Indicates whether data should be checked to make sure it has not been read or intercepted. Default: not_required
msg_sequence_check	Indicates whether data should be checked for interception. Default: not_required
mutual_auth	Requires remote server to provide proof of identify before a connection is established. Default: not_required
security_mechanism	The name of the third-party security mechanism enabled for the pathway. Default: first mechanism listed in the SECURITY section of *libtcl.cfg*
unified_login	Indicates how Replication Server seeks to log in to remote data servers and accepts incoming logins. The values are: • required – always seeks to log in to remote server with a credential. • not_required – always seeks to log in to remote server with a password. Default: not_required
use_security_services	Tells Replication Server whether to use security services. If use_security_services is “off,” no security features take effect. **Note** This parameter can only be set by configure replication server.

Examples

Example 1 Entered at the TOKYO_RS Replication Server, this command creates a direct route from TOKYO_RS to the SYDNEY_RS Replication Server. TOKYO_RS can log in to SYDNEY_RS over this route, using the login name “sydney_rsi” with the password “sydney_rsi_ps:”

```
create route to SYDNEY_RS
 set username sydney_rsi
 set password sydney_rsi_ps
```

Example 2 Entered at TOKYO_RS, this command creates an indirect route from TOKYO_RS to SYDNEY_RS, through the intermediate Replication Server, MANILA_RS. Direct routes must already exist from TOKYO_RS to MANILA_RS and from MANILA_RS to SYDNEY_RS:

```
create route to SYDNEY_RS
 set next site MANILA_RS
```

Example 3 This command creates a direct route similar to that in the first example. However, if network-based security is enabled, TOKYO_RS must log in to SYDNEY_RS with a credential:

```
create route to SYDNEY_RS
 set unified_login 'required'
```

Usage

- Use create route to create a direct or indirect route from the current Replication Server to a remote Replication Server.
- Before creating a route, you should have determined your overall routing scheme. See the *Replication Server Administration Guide Volume 1* for information on creating and managing routes.
- Replication Server does not support routing schemes where routes diverge from the same source Replication Server, then converge at the same intermediate or destination Replication Server.
- Replication Server distributes information about the new route to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.
- If Replication Server is configured with Embedded RSSD (ERSSD), you can create a route as long as both Replication Servers are 15.0 or higher. If the route being created is the first route originating from the current site, log transfer will be started and a Replication Agent will be started automatically:

  To change the Replication Agent’s name, enter:

  ```
  configure replication server
  set erssd_ra to 'value'
  ```

Direct routes

- Specify an RSI user name and password and omit the next site clause from create route to set up a direct route from the current Replication Server to the destination Replication Server.
- Before you create a direct route, create login names and passwords in the destination Replication Server. You can use rs_init to set these up; the default user name is “*RS_name*_rsi” and the default password is “*RS_name*_rsi_ps.”

  If a route is created with a user and password that do not exist at the destination Replication Server, add or change the user and password at that destination.

Indirect routes

- Include the next site clause in create route to set up an indirect route for Replication Server messages. For example, messages originating in New York and destined for all European sites can be routed through a London site, along an indirect route. Using indirect routes decreases the volume of messages passed through a portion of the route.
- Before you create an indirect route, you must first create a direct route from the source Replication Server to the intermediate Replication Server, and from the intermediate Replication Server to the destination Replication Server.
- A route can have any number of intermediate Replication Servers. However, because each additional intermediate Replication Server increases the lag time between the primary and replicate sites, you should limit the number of intermediate sites.

Routes and RSSD tables

- The RSI user name and password you specify when you create a direct route is added to the rs_users system table in the RSSD of the destination Replication Server. The user name and password are also added to the rs_maintusers system table in the RSSD of the source Replication Server.
- When you create a route, the source Replication Server sends the destination Replication Server the login name and password of the source RSSD's primary user. The destination Replication Server uses this login to create subscriptions to some of the RSSD system tables at the source Replication Server. This primary user login name is usually named "*source_RSSD_name*_prim," and is stored in the rs_users system table at the destination Replication Server.

Network-based security parameters

- Both ends of a route must use compatible Security Control Layer (SCL) drivers with the same security mechanisms and security features. It is the replication system Administrator's responsibility to choose and set security features for each server. The Replication Server does not query the security features of remote servers before it attempts to establish a connection.
- create route specifies network-based security settings that affect how the current Replication Server logs in to the target Replication Server and how secure message transmission is accomplished.

- If unified_login is set to "required," *only* the "sa" user can log in to the Replication Server without a credential. If the security mechanism should fail, the "sa" user can then log in to Replication Server with a password and disable unified_login.
- A Replication Server can have more than one security mechanism; each supported mechanism is listed in the *libtcl.cfg* file under SECURITY.
- Message encryption is a costly process with severe performance penalties. In most instances, it may be wise to set msg_confidentiality "on" only for certain routes. Alternatively, choose a less costly security feature, such as msg_integrity.

Permissions

create route requires "sa" permission.

See also

alter connection, alter route, configure replication server, create connection, drop connection, drop route

create subscription

Description

Creates and initializes a subscription and materializes subscription data. The subscription may be for a database replication definition, table replication definition, function replication definition, or publication.

Syntax

```
create subscription sub_name
for { table_repdef | func_repdef | { publication pub |
     database replication definition db_repdef  }
     with primary at server_name.db  }
with replicate at data_server.database
[where {column_name | @param_name}
     {< | > | >= | <= | = | &} value
[and {column_name | @param_name}
     {< | > | >= | <= | = | &} value]...]
[without holdlock | incrementally | without materialization]
[subscribe to truncate table]
[for new articles]
```

Parameters

sub_name
: The name of the subscription, which must conform to the rules for naming identifiers. The subscription name must be unique for the replication definition, where applicable, and for the replicate data server and database.

for *table_rep_def*
: Specifies the table replication definition the subscription is for.

for *function_rep_def*
: Specifies the name of the function replication definition the subscription is for.

for publication *pub_name*
: Specifies the publication the subscription is for.

for database replication definition *db_repdef*
: Specifies the database replication definition the subscription is for.

with primary at *data_server*.*database*
: Include this clause for a subscription for a publication or a database replication definition. Specifies the location of the primary data. If the primary database is part of a warm standby application that uses logical connections, *data_server.database* is the name of the logical data server and database.

with replicate at *data_server.database*
: Specifies the location of the replicate data. If the replicate database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

where

Sets criteria for the column or parameter values that are to be replicated via the subscription. If you omit the where clause, all rows or parameters are replicated.

You can include a where clause in a subscription for a table or function replication definition. You cannot include a where clause in a database or publication subscription.

A where clause is composed of one or more simple comparisons, in which a searchable column or searchable parameter from the replication definition is compared to a literal value using one of these relational operators: <, >, <=, >=, =, or &. (The & operator is supported *only* for rs_address columns or parameters.) You can join comparisons using the keyword and.

Column or parameter names used in the expression must be included in the searchable columns list of the table replication definition or the searchable parameters list of the function replication definition.

Java columns cannot be evaluated in subscription expressions. Thus, you cannot include a Java column of type rawobject or rawobject in row in a where clause.

The maximum size of a where clause in a subscription is 255 characters.

Note You cannot convert binaries with less than seven bytes into integers.Workarounds include using zeros to pad binary values up to eight bytes, or using integer values instead of binary values.

column_name

A column name from the primary table, for a subscription to a table replication definition.

@param_name

A parameter name from a replicated stored procedure, for a subscription to a function replication definition.

value

A value for a specified column or parameter. See “Datatypes” on page 21 for entry formats for values for different datatypes.

Column or parameter names used in the expression must be included in the searchable columns or searchable parameters list of the replication definition.

without holdlock
: Selects data from the primary database without a holdlock, for non-atomic materialization. The rows are applied at the replicate database in increments of 10-row inserts per transaction. See "Nonatomic materialization" on page 252 for more information.

incrementally
: Initializes the subscription and apply subscription data in increments of 10-row inserts per transaction. A holdlock is used on the primary database, for atomic materialization.

without materialization
: Does not materialize data for the subscription. Use this option when there is no activity at the primary database and the data already exists in the replicate database. Or, use this option when you have suspended activity in the primary database and manually transferred the data to the replicate database. Database subscriptions must include this option.

subscribe to truncate table
: For a subscription to a table replication definition, a database replication definition, or to a publication, enables replication of the truncate table command to the subscribing replicate database.

  You must set this option the same as it is set for any existing subscriptions that replicate data into the same replicate table for a particular database. Otherwise, the new subscription is rejected.

for new articles
: Refreshes an existing subscription. Instructs Replication Server to check the subscription against the publication and then to create subscriptions against unsubscribed articles.

Examples

Example 1 Creates a subscription named titles_sub. It specifies that rows from the titles table with columns of the type "business" are to be replicated in the titles table in the pubs2 database of the data server named SYDNEY_DS:

```
create subscription titles_sub
 for titles_rep
 with replicate at SYDNEY_DS.pubs2
 where type = 'business'
```

Example 2 Creates a subscription named titles_sub that includes rows from the titles table with prices that are greater than or equal to $10.00:

```
create subscription titles_sub
 for titles_rep
 with replicate at SYDNEY_DS.pubs2
 where price >= $10.00
```

Example 3 Creates a subscription named myproc_sub for the function replication definition myproc_rep. In order to use this command to create a subscription for a function replication definition, data must already exist at the replicate database, and you must use the without materialization clause:

```
create subscription myproc_sub
 for myproc_rep
 with replicate at SYDNEY_DS.pubs2
 without materialization
```

Example 4 Creates a subscription named pubs2_sub for the publication pubs2_pub:

```
create subscription pubs2_sub
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replicate at SYDNEY_DS.pubs2
```

Example 5 Creates a database subscription pubs2_sub for the database replication definition pubs2_rep:

```
create subscription pubs2_sub
 for database replication definition pubs2_rep
   with primary at NEWYORK_DS.pubs2
   with replicate at TOKYO_DS.pubs2
 without materialization
 subscribe to truncate table
```

Usage

- To subscribe to a function or database replication definition, use create subscription with the without materialization clause, or use define subscription and the other bulk materialization commands.
- Execute create subscription at the Replication Server of the database where the replicated data will be stored.
- See the *Replication Server Administration Guide Volume 1*, for more information about subscriptions and the role they play in replication.
- If you need to change which replication definition a subscription is for, you must drop the subscription and re-create it, specifying the name of the replication definition to which you want to subscribe.

- You can create multiple replication definitions for the same primary table or database. You cannot subscribe to more than one replication definition for the same replicate table or database, although you can subscribe to the same replication definition more than once.
- If you want to materialize text, unitext, image, or rawobject data, you can use automatic materialization *only* if the size of your data row is less than 32K. Otherwise, you must use bulk materialization.

Subscribing to database replication definitions

- When you create a database subscription, you cannot use the where clause to limit data subscription. All data is subscribed.
- With database subscriptions, you can use only the no materialization or bulk materialization methods. Use define subscription to use dump and load or other bulk materialization method. Use create subscription to use the no materialization method.
- You cannot subscribe to more than one database replication definition from the same origin.

Subscribing to publications

- When a publication is valid, you can create a subscription for the publication in order to begin replication to a replicate database. All forms of subscription materialization are supported.
- When you create a publication subscription, Replication Server creates a separate underlying subscription for each article that the publication contains. Each article subscription uses the name of the parent publication subscription.
  - When you use atomic or non-atomic materialization, article subscriptions are materialized one at a time in the order that the articles were added to the publication.
  - When you use create subscription with the without materialization clause, all article subscriptions are activated and validated at the same time.
- A subscription to a publication cannot include a where clause. Instead, you can customize replication to replicate sites by including one or more where clauses in each article the publication contains.

Specifying columns subject to HDS translations

- When you create a subscription that includes a where clause, make sure that the value in the where clause comparison is in the declared datatype format.

- Subscriptions that specify columns subject to class- or column-level translations in the where clause cannot be dematerialized automatically. You must use either the bulk or the no-materialization method.

Replicating *truncate table*

- When you create the first subscription, you can either include or not include the subscribe to truncate table option. Each subsequent subscription that replicates into the same table must follow the example of the first subscription. Otherwise, the subscription is rejected when you try to create it.
- You can change the current "subscribe to truncate table" status of a particular replicate table by executing sysadmin apply_truncate_table.

Requirements for executing *create subscription*

- In addition to the permissions listed below, make sure that these requirements are met before you execute create subscription.

  For a subscription to a table replication definition:

  - A replication definition exists for the primary table you are replicating, and the table is marked for replication with sp_setreptable.
  - If you subscribe to tables marked using sp_reptostandby, you must configure the primary database connection using the rep_as_standby configuration parameter and configure RepAgent using send_warm_standby_exacts.
  - Tables referenced in the replication definition exist in both the primary and the replicate database. Each table has the columns and datatypes defined in the replication definition.

    This table is visible to the user creating the subscription and to the user maintaining it. The easiest way to achieve this is to have the Database Owner create the table.

  For a subscription to a function replication definition:

  - A replication definition exists for the stored procedure you are replicating, and the stored procedure is marked for replication with sp_setrepproc.
  - Stored procedures referenced in the function replication definition exist in both the primary and replicate database. Each stored procedure has the parameters and datatypes defined in the function replication definition.

For a subscription to a publication:

- A publication exists that contains articles for the primary tables or stored procedure you are replicating. The articles specify replication definitions that meet the requirements described above.
- The publication is valid.

Requirements for warm standby applications

- These requirements apply when you create subscriptions in warm standby applications:
  - If the destination database is part of a warm standby application, the table must exist in both the active and standby databases. Both tables must be marked for replication using sp_setreptable or sp_reptostandby.
  - For a logical primary database, you cannot create a subscription while Replication Server is in the process of adding a standby database.

Requirements for tables with the same name

- If a primary Adaptive Server database contains a replicated table and another table that has the same name, the owner of the second (unreplicated) table cannot create a subscription to the replicated table without using custom rs_select or rs_select_with_lock function strings. For example:
  - If there is a replication definition for a primary table named db.dbo.table1, and
  - Database user "jane" owns a table named db.jane.table1, then
  - Jane cannot create a subscription to the replication definition for db.dbo.table1 using the default function strings.

Atomic materialization

- The default method for materializing subscriptions with this command is atomic materialization. Atomic materialization locks the primary table and copies subscription data through the network in a single atomic operation.
- During atomic materialization, no rows appear at the replicate database until the select transaction has been completed in the primary database. If the subscription specifies a large number of rows, the select transaction can run for a long time, causing a delay at the replicate site.

Requirements for using atomic materialization

- If you plan to use the atomic method of subscription materialization:
  - You or the Database Owner must own the primary table, or you must use user-defined function strings for select operations at the primary database.

- The Database Owner or the maintenance user must own the replicate table, or you must use user-defined function strings for select operations at the replicate database. If the owner of the replicate table is different from the owner of the primary table, you must create a unique function string by using a distinct function-string class.

Using the *without holdlock* or *incrementally* option

- The without holdlock or incrementally options are alternatives to the default atomic method of subscription materialization. When you specify these options, Replication Server applies the rows in batches, so that data appears at the replicate database a batch at a time.

  As a result, during materialization, queries at the replicate database may return incomplete data for the subscription. This temporary condition ends when check subscription indicates the subscription is valid.

The *incrementally* option

- The incrementally option is a variation of atomic materialization. Use this option for large subscriptions to avoid a long-running transaction at the replicate database. The subscription data is not applied atomically at the replicate database, so the data is available; however, it is incomplete until materialization has completed and the subscription is validated.
- When incrementally is used, the select is performed with a holdlock to maintain serial consistency with the primary database. The replicate table passes through states that occurred previously at the primary database.

  In all cases, replicate data is consistent with the primary database by the time materialization completes and check subscription indicates that the subscription is valid.

Nonatomic materialization

- The without holdlock option uses non-atomic materialization. When this option is specified, materialization rows are selected from the primary database without a holdlock. This can introduce inconsistency if rows are updated at the primary database after the select. To correct inconsistencies, use set autocorrection on when using without holdlock.
- When data already exists at the replicate database, you can use atomic or non-atomic materialization instead of bulk materialization.

Requirements for using nonatomic materialization

- If you plan to use non-atomic method of subscription materialization:
  - Do not use without holdlock if you update data by distributing applied functions from the primary database or if you update the data with commutative functions. For example, if a stored procedure updates a row by incrementing the previous value of a column, the value may be incorrect when materialization has completed.

- If the replicate minimal columns option is set for a replication definition, you cannot use without holdlock to create new subscriptions.
- For non-atomic subscriptions, if a non-atomic subscription is materializing when switch active executes, it is marked “SUSPECT.

No materialization

The without materialization clause specifies the no-materialization method. It provides an convenient way to create a subscription when the subscription data already exists at the replicate database.

Requirements for no materialization

- The subscription data must already exist at the replicate database.
- The primary and replicate database must be in sync.
- Activity must be stopped at the primary database so that there are no further updates in the Replication Server stable queue.

Using the *rs_address* datatype

- You can subscribe to replication definitions whose columns or parameters use the special datatype rs_address. This datatype allows a unique subscription resolution method, whereby bitmaps of the rs_address datatype (based on the underlying int datatype) are compared with a bitmask in a subscription’s where clause. The bitmap comparison tells the primary Replication Server whether or not a replicate site should receive the data in each row.
- For rs_address columns or parameters *only*, the bitmap comparison operator & is supported in the where clause, as follows:

```
where rs_address_column1  & bitmask
[and rs_address_column2  & bitmask]
 [and other_search_conditions]
```

- Replication Server does not replicate a row if the only changed columns are rs_address columns, unless the changed bits indicate that the row should be inserted or deleted at the replicate database.

  Because of this filtering, rs_address columns in replicate databases may not be identical to the corresponding columns at the primary database. This optimizes applications that use rs_address columns to specify the destination replicate databases.

How the *rs_address* datatype works

- Each bit in an rs_address column field may represent a category of data, such as inventory or billing. In a subscription bitmask, you set the corresponding bit to “on” (1), for each category of data you want to replicate to the subscribing site.

For example, users at a warehouse site who are interested in inventory data would set the inventory bit to "on" in a subscription bitmap. If the same warehouse users are not interested in billing data, they would set that bit to "off" (0). When a bit is set to "on" in both a subscription bitmask and an rs_address column, the row containing the bit is replicated.

32-bit limitation of underlying *int* datatype for *rs_address*

- Due to the 32-bit limitation of the underlying int datatype, you may need to construct primary tables with more than one rs_address column. The and keyword allows you to create a single subscription to perform bitmap comparisons on more than one rs_address column.

  However, to subscribe to a row when one or more bits are set in either of two or more rs_address columns, you must create separate subscriptions.

Using 32-bit hexadecimal numbers for *rs_address*

- You can also specify search conditions for non-rs_address columns using the and keyword and the comparison operators (other than &) described in the command syntax. If you use and to specify search conditions, subscription data may not be replicated or may migrate out of a subscription, even if rs_address bitmap comparisons would otherwise replicate a row.
- You can compare rs_address columns to 32-bit integer values or 32-bit hexadecimal numbers in the where clause. If you use hexadecimal numbers, pad each number with zeros, as necessary, to create an 8-digit hexadecimal value.

Warning! Be very cautious about comparing rs_address columns to hexadecimal numbers in the where clause of a subscription. Hexadecimal values are treated as binary strings by Adaptive Server and Replication Server. Binary strings are converted to integers by copying bytes. The resulting bit pattern may represent different integer values on different platforms.

For example, 0x0000100 represents 65,536 on platforms that consider byte 0 most significant, and represents 256 on platforms that consider byte 0 least significant. Because of these byte-ordering differences, bitmap subscriptions involving hexadecimal numbers may not work in a multi-platform replication system.

- See "Datatypes" on page 21 for more information about the rs_address and int datatypes. Also, see the *Replication Server Administration Guide Volume 1*.

- Refer to the *Adaptive Server Enterprise Reference Manual* and the *Open Client and Open Server Common Libraries Reference Manual* for more information about conversion between datatypes.

Monitoring a subscription

- When Replication Server materializes a subscription, it logs in to the primary data server, using the subscription creator's login name, and selects the rows from the primary table. Use check subscription to monitor the progress of the materialization.
- create subscription returns a prompt before the data materialization is complete. Materialization is complete when check subscription reports "VALID" at the replicate Replication Server.

Permissions

To execute create subscription, you must have the following login names and permissions:

- The same login name and password at the replicate Replication Server, primary Replication Server, and primary Adaptive Server database.
- "create object" or "sa" permission at the replicate Replication Server where you enter this command.
- "create object", "primary subscribe", or "sa" permission at the primary Replication Server.
- select permission on the primary table in the primary Adaptive Server database.
- execute permission on the rs_marker stored procedure in the primary Adaptive Server database.
- The replicate database maintenance user must have select, insert, update, and delete permissions on the replicate table, and execute permissions for functions used in replication.

See also

alter database replication definition, alter replication definition, check subscription, create article, create database replication definition, create function replication definition, create function string, create publication, create replication definition, define subscription, drop subscription, set autocorrection, sysadmin apply_truncate_table

create user

Description	Adds a new user login name to a Replication Server.
Syntax	create user *user* set password {*passwd* | null}
Parameters	*user* The new user's Replication Server login name. Login names must conform to the rules for identifiers. *passwd* The user's password. It can be up to 30 characters long and can include letters, numerals, and symbols. Case is significant. If the password contains spaces, enclose the password in quotation marks. When you create or alter a user login name, you must specify a password or "null." A null password lets a user log in without being prompted for a password.

Examples	Creates a new user login name "louise" with the password "EnnuI":
	`create user louise`
	` set password EnnuI`

Usage	• create user creates a new login name for a user. • Users can change their own passwords with the alter user command. • Case is significant for user login names and passwords.
Permissions	create user requires "sa" permission.
See also	alter user, drop user, grant, revoke

define subscription

Description

Adds a subscription to the Replication Server system tables, but does not materialize or activate the subscription. The subscription may be for a database replication definition, a table replication definition, a function replication definition, or for a publication. This command begins the process of bulk subscription materialization, or the process of refreshing a publication subscription.

Syntax

```
define subscription sub_name
for {table_rep_def | function_rep_def |
    publication pub_name | database replication definition db_repdef
    with primary at data_server.database } |
with replicate at data_server.database
    [where {column_name | @param_name}
    {< | > | >= | <= | = | &} value
    [and {column_name | @param_name}
    {< | > | >= | <= | = | &} value]...]
[subscribe to truncate table]
[for new articles]
[use dump marker]
```

Parameters

sub_name
The name of the subscription, which must conform to the rules for naming identifiers. The subscription name must be unique for the replication definition, where applicable, and for the replicate data server and database.

for *table_rep_def*
Specifies the table replication definition the subscription is for.

for *function_rep_def*
Specifies the name of the function replication definition the subscription is for.

for publication *pub_name*
Specifies the publication the subscription is for.

for database replication definition *db_repdef*
Specifies the database replication definition the subscription is for.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database. Include this clause only for a subscription for a publication.

with replicate at *data_server.database*

Specifies the location of the replicate data. If the replicate database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

where

Sets criteria for the column or parameter values that are to be replicated via the subscription. If you omit the where clause, all rows or parameters are replicated.

You can include a where clause in a subscription for a table or function replication definition. You cannot include a where clause in a publication subscription.

A where clause is composed of one or more simple comparisons, in which a searchable column or searchable parameter from the replication definition is compared to a literal value using one of these relational operators: <, >, <=, >=, =, or &. (The & operator is supported *only* for rs_address columns or parameters.) You can join comparisons with the keyword and.

Column or parameter names used in the expression must be included in the searchable columns list of the table replication definition or in the searchable parameters list of the function replication definition.

Java columns cannot be evaluated in subscription expressions. Thus, you cannot include a Java column of type rawobject or rawobject in row in a where clause.

column_name

A column name from the primary table, for a subscription to a table replication definition.

@param_name

A parameter name from a replicated stored procedure, for a subscription to a function replication definition.

value

A value for a specified column or parameter. See “Datatypes” on page 21 for entry formats for values for different datatypes.

subscribe to truncate table

For a subscription to a table replication definition or to a publication, enables replication of the truncate table command to the subscribing replicate database.

You must set this option the same as it is set for any existing subscriptions that replicate data into the same replicate table. Otherwise, the new subscription will be rejected.

for new articles

Refreshes an existing subscription. Instructs Replication Server to check the subscription against the publication and then to create subscriptions against unsubscribed articles.

use dump marker

Tells Replication Server to apply transactions to a replicate database. use dump marker activates and validates the database subscription automatically. Without this option, users must activate and validate the database subscription manually.

Note Use dump marker one at a time as you cannot define multiple database subscriptions with dump marker. You also need to place a dump database command between each subscription command.

Examples

Example 1 Creates a subscription named titles_sub. It specifies that rows from the titles table with columns of the type "business" are to be replicated in the titles table in the pubs2 database of the data server named SYDNEY_DS:

```
define subscription titles_sub
  for titles_rep
    with replicate at SYDNEY_DS.pubs2
    where type = 'business'
```

Example 2 Creates a subscription named titles_sub that includes rows from the titles table with prices that are greater than or equal to $10.00:

```
define subscription titles_sub
  for titles_rep
    with replicate at SYDNEY_DS.pubs2
    where price >= $10.00
```

Example 3 Creates a subscription named myproc_sub for the function replication definition myproc_rep:

```
define subscription myproc_sub
```

```
        for myproc_rep
          with replicate at SYDNEY_DS.pubs2
```

Example 4 Creates a subscription named pubs2_sub for the publication pubs2_pub:

```
define subscription pubs2_sub
  for publication pubs2_pub
    with primary at TOKYO_DS.pubs2
    with replicate at SYDNEY_DS.pubs2
```

Example 5 Creates a subscription pubs2_sub for the database replication definition pubs2_rep:

```
define subscription pubs2_sub
  for database replication definition pubs2_rep
    with primary at NEWYORK_DS.pubs2
    with replicate at TOKYO_DS.pubs2
    subscribe to truncate table
    use dump marker
```

Refer to the *Replication Server Design Guide* for examples of creating subscriptions for a complete replication system.

Usage

- Use define subscription to create a subscription manually using bulk materialization. With bulk materialization, subscription creation and materialization is performed in discrete steps so that you can load the initial data from media rather than sending it from the primary database through the WAN.
- If you have added any new articles to a publication with an existing subscription, you must refresh the publication subscription in order to create new subscriptions for these articles.
- Activate the subscription using activate subscription and validate the subscription using validate subscription.
- Although you can create multiple replication definitions for the same primary table, you cannot subscribe to more than one replication definition for the same replicate table. However, you can subscribe to the same replication definition more than once.

Subscribing to publications

- You can create a subscription for a valid publication to begin replication to a replicate database. All forms of subscription materialization are supported.

- Use define subscription to create new article subscriptions in the publication subscription. Then use activate subscription, manually load the subscription data for the new article subscriptions, and use validate subscription to validate the publication subscription.
- When you create a publication subscription, Replication Server creates a separate underlying subscription for each article that the publication contains. Each article subscription uses the name of the parent publication subscription.
- When you activate and validate a publication subscription, all of its article subscriptions are activated and validated at the same time.
- A subscription to a publication cannot include a where clause. Instead, you can customize replication to replicate sites by including one or more where clauses in each article the publication contains.

Subscribing to database replication definitions

- When you create a database subscription, you cannot use the where clause to limit data subscription. All data is subscribed.
- With database subscriptions, you can use only the no materialization or bulk materialization methods. Use define subscription to use dump and load or other bulk materialization method. Use create subscription to use the no materialization method.
- You cannot subscribe to more than one database replication definition from the same origin.

Replicating *truncate table*

- When you create the first subscription for a table, you can either include or not include the subscribe to truncate table option. Each subsequent subscription that copies information into the same table must follow the example of the first subscription. Otherwise, it will be rejected when you try to create it.
- You can view or change the current “subscribe to truncate table” status of a particular replicate table by executing sysadmin apply_truncate_status.

Working with the *rs_address* datatype

See create subscription for information about working with columns or parameters that use the rs_address datatype.

Requirements for executing *define subscription*

- In addition to the permissions listed below, make sure these requirements are met before you execute this command.

- For a subscription to a table replication definition:
  - A replication definition exists for the primary table you are replicating, and the table is marked for replication with sp_setreptable.
  - Tables referenced in the replication definition exist in both the primary and the replicate database. Each table has the columns and datatypes defined in the replication definition.

    This table is also visible to the user creating the subscription and the user maintaining it. The easiest way to achieve this is to have the Database Owner create the table.

  For a subscription to a function replication definition:

  - A replication definition exists for the stored procedure you are replicating, and the stored procedure is marked for replication with sp_setrepproc.
  - Stored procedures referenced in the function replication definition exist in both the primary and replicate database. Each table has the parameters and datatypes defined in the function replication definition.

  For a subscription to a publication:

  - A publication exists that contains articles for the primary tables or stored procedure you are replicating. The articles specify replication definitions that meet the requirements described above.
  - The publication is valid.

Creating subscriptions using *define subscription*

- You can use define subscription to subscribe to a table replication definition, a function replication definition, or a publication.
  - For a subscription to a table replication definition, enter define subscription at the Replication Server that manages the database where the replicate data is to be stored.
  - For a subscription to a function replication definition, enter define subscription at the Replication Server that manages the database where the destination stored procedure is to be executed via applied function delivery.

    - For a subscription to a publication, enter define subscription at the Replication Server that manages the database where the replicate data is to be stored or where destination stored procedures are to be executed.

- A table subscription maintains a replicate copy of a table, or selected rows from a table, in a database. Changes made to the primary version are also applied to the copy.
- A function subscription replicates user-defined function invocations associated with a function replication definition. A replicated function typically includes parameters and modifies data, but it need not involve replicated data.
- A publication subscription involves underlying subscriptions for the articles the publication contains, which replicate table or user-defined function invocations depending on the replication definitions in the article.
- See the *Replication Server Administration Guide Volume 1* for more information about subscriptions and the role they play in replication.

Alternative command to create subscriptions

- Use create subscription to create, materialize, activate, and validate, in a single step, a subscription for a table replication definition, function definition replication, or publication.

Permissions

To execute define subscription, you must have the following login names and permissions:

- The same login name and password at the replicate Replication Server, primary Replication Server, and primary database.
- “create object” or “sa” permission at the replicate Replication Server where you enter this command.”
- “create object”, “primary subscribe”, or “sa” permission at the primary Replication Server.

See also

alter replication definition, activate subscription, check subscription, create article, create function replication definition, create publication, create replication definition, create subscription, drop subscription, sysadmin apply_truncate_table, validate subscription

drop article

Description

Drops an article and optionally drops its replication definition.

Syntax

drop article *article_name*
for *pub_name*
with primary at *data_server.database*
[drop_repdef]

Parameters

article_name
The name of the article to drop.

for pub_name
Specifies the name of the publication the article is for.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

drop_repdef
An optional keyword that causes the replication definition the article is for to be dropped—if it is not used elsewhere.

Examples

Example 1 Drops the article named titles_art for the publication pubs2_pub in the TOKYO_DS.pubs2 database:

```
drop article titles_art
 for pubs2_pub
 with primary at TOKYO_DS.pubs2
```

Example 2 Drops the article named titles_art for the publication pubs2_pub in the TOKYO_DS.pubs2 database. This command also drops the replication definition the article is for, if it is not used elsewhere:

```
drop article titles_art
 for pubs2_pub
 with primary at TOKYO_DS.pubs2
 drop_repdef
```

Usage

- Use drop article to remove an article from a publication. Execute drop article at the Replication Server that manages the database where the primary data is stored.
- You can drop an article if there are no subscriptions for the article. Drop subscriptions first, as necessary.

- Optionally, you can also drop the replication definition for the article, if it is not part of any other article and has no subscriptions.
- A dropped article is removed at the replicate site only when create/define subscription is executed there.

Dropping articles from a publication with a subscription

- If you drop an article from an existing publication, the publication is invalidated. You must drop all existing article subscriptions using drop subscription for article before the article can be dropped. To create new publication subscriptions you must:
  - Validate the publication when you have completed making changes to the publication, then

  See create subscription and define subscription for more information on the two methods of refreshing publication subscriptions.

Permissions

drop article requires “create object” permission.

See also

check subscription, create article, create publication, create subscription, define subscription, drop function replication definition, drop publication, drop replication definition, drop subscription

drop connection

Description

Removes a database from the replication system.

Syntax

drop connection to *data_server*.*database*

Parameters

data_server
The name of the data server with the database to be removed from the replication system.

database
The name of the database whose connection is to be dropped.

Examples

Drops the connection to the pubs2 database in the SYDNEY_DS data server:

```
drop connection to SYDNEY_DS.pubs2
```

Usage

- Use drop connection to remove database connection information from the Replication Server system tables. This command does not remove replicated data from any database in the system.
- Before you drop a connection:
  - Drop any subscriptions that replicate data to the database.
  - If the connection is to a primary database, drop any replication definitions for tables in the database.
- Before you re-create a connection to a database with the same name, you may need to use sysadmin dropdb.
- Replication Server distributes information about the dropped database connection to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

Permissions

drop connection requires “sa” permission.

See also

alter connection, create connection, resume connection, suspend connection, sysadmin dropdb

drop database replication definition

Description	Deletes an existing database replication definition.

Syntax

drop database replication definition *db_repdef*
  with primary at *server_name.db*

Parameters

db_repdef
  Name of the database replication definition.

server_name.db
  Name of the primary server/database combination. For example: TOKYO.dbase.

Examples

Deletes the database replication definition dbrep1:

```
drop database replication definition dbrep1
  with primary at PDS.my_db
```

Usage

drop database replication definition succeeds only if there is no database subscription to the named database replication definition.

See also

alter database replication definition, create database replication definition

drop error class

Description

Drops an error class and any actions associated with it.

Syntax

drop error class *error_class*

Parameters

error_class
The name of the error class to drop.

Examples

Drops the pubs2_db_err_class error class from the Replication Server. Also drops any error actions that were assigned for the pubs2_db_err_class error class:

```
drop error class pubs2_db_err_class
```

Usage

- Use the drop error class command to remove an error class. When an error class is dropped, all actions assigned for it are also dropped.
- You execute drop error class at the Replication Server where the error class was created.
- You cannot drop:
    - The rs_sqlserver_error_class error class.
    - An error class that is in use with a database
- To change the primary site for an error class, use the move primary of error class command.
- Replication Server distributes information about the dropped class to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system latency.

Permissions

drop error class requires “sa” permission.

See also

assign action, create connection, create error class, drop connection, move primary

drop function

Description	Drops a user-defined function and its function strings.
Syntax	drop function [*replication_definition*.]*function*

Parameters

replication_definition
The name of the replication definition the function was created for.

function
The name of the function to drop.

Examples

Drops the upd_publishers user-defined function for the publishers_rep replication definition. Also drops any function strings defined for the function:

```
drop function publishers_rep.upd_publishers
```

Usage

- Use drop function to remove a function name and any function strings that have been created for it.
- Execute drop function at the Replication Server where the replication definition was created.
- You cannot drop system functions. For more information about system functions, see Chapter 4, “Replication Server System Functions.”
- Replication Server distributes information about the dropped user-defined function to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.
- When you drop a user-defined function for a replication definition, it is dropped for all replication definitions in the primary table.
- Do not execute drop function for replicated functions. Use drop function rep def instead.

Permissions

drop function requires “create object” permission.

See also

create function, drop function string, move primary

drop function replication definition

Description

Drops a function replication definition and its user-defined function.

Syntax

drop function replication definition *function_rep_def*

Parameters

function_rep_def
The name of the function replication definition to drop.

Examples

Drops the function replication definition named titles_frep and its user-defined function and function string:

```
drop function replication definition titles_frep
```

Usage

- Use drop function replication definition to remove a function replication definition.
- Before you can drop a function replication definition, you must drop all subscriptions for it.
- Execute drop function replication definition at the primary Replication Server for the function replication definition.
- After you drop the stored procedure defined by this function replication definition, execute sp_setrepproc in the database, setting the procedure's replicate status to 'false'. This stops the RepAgent from transferring log entries to the Replication Server.
- Replication Server distributes information about the dropped function replication definition to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

Permissions

drop function replication definition requires "create object" permission.

See also

alter function replication definition, check subscription, create function replication definition, create subscription, define subscription, drop subscription

drop function string

Description

Drops a function string for a function-string class.

Syntax

drop function string
 [*replication_definition*.]*function*[;*function_string* | all]
 for *function_class*

Parameters

replication_definition
The name of the table or function replication definition the function operates on.

function
The name of the function the function string was created for.

function_string
The name of the function string to drop. The default function string name is the same as the function name.

all
Causes Replication Server to drop all function strings for a function. Although only the rs_select, rs_select_with_lock, rs_datarow_for_writetext, rs_get_textptr, rs_textptr_init, and rs_writetext functions can have multiple function strings, this option can be used as shorthand for the *function_string* name.

function_class
The name of the function-string class from which the function string will be dropped.

Examples

Example 1 Drops the function strings for the rs_insert function for the publishers_rep replication definition in the derived class sqlserver_derived_class. The rs_insert function string will now be inherited from the parent class:

```
drop function string
 publishers_rep.rs_insert
 for sqlserver_derived_class
```

Example 2 Drops the function string for the upd_publishers user-defined function for the publishers_rep replication definition in the sqlserver2_function_class function-string class:

```
drop function string
 publishers_rep.upd_publishers
 for sqlserver2_function_class
```

Example 3 Drops all function strings for the rs_select_with_lock function for the publishers_rep replication definition in the class sqlserver2_func_class:

```
drop function string
 publishers_rep.rs_select_with_lock;all
 for sqlserver2_func_class
```

Usage

- To replace an existing function string with a new one, use either alter function string or create function with overwrite.

Warning! If a transaction occurs between the time a function string is dropped and the time it is re-created, Replication Server detects the function string as missing and fails the transaction.

- Dropping a function drops corresponding function strings from all function-string classes.
- Dropping a customized function string from a derived function-string class causes that class to inherit the function-string from its parent class.
- Dropping a customized function string from rs_sqlserver_function_class causes Replication Server to delete the customized and default function string. To revert the customized function string to the default function string for a function in the rs_sqlserver_function_class, use alter function string and omit the output clause.
- Replication Server distributes information about the dropped function string to qualifying sites through the replication system. The changes do not appear immediately at all such sites because of normal replication system lag time.

Permissions

drop function string requires “create object” permission.

See also

alter function string, create function, create function string, create function string class, drop function

drop function string class

Description

Drops a function-string class.

Syntax

drop function string class *function_class*

Parameters

function_class
The name of the function-string class to drop.

Examples

Example 1 Drops the derived function-string class sqlserver_derived_class and all of its customized function strings:

```
drop function string class
 sqlserver_derived_class
```

Example 2 Drops the function-string class sqlserver2_function_class and its function strings:

```
drop function string class
 sqlserver2_function_class
```

Usage

- Use drop function string class to remove a function-string class. function-string classes group all function strings for a database.
- Dropping a function-string class also drops all of the associated function strings and removes all references to the class.
- A function-string class that is still in use on a database connection cannot be dropped.
- You cannot drop any of the three system-provided classes, rs_sqlserver_function_class, rs_default_function_class, or rs_db2_function_class.
- You cannot drop any function-string class that is a parent class for an derived class.

Permissions

drop function string class requires “sa” permission.

See also

create function string class, drop function, drop function string

drop logical connection

Description

Drops a logical connection. Logical connections are used to manage warm standby applications.

Syntax

drop logical connection to *data_server.database*

Parameters

data_server
The logical data server specified in the create logical connection command.

database
The name of the database specified in the create logical connection command.

Examples

Drops the logical connection for a data server named LDS and a database named *pubs2*:

```
drop logical connection to LDS.pubs2
```

Usage

- Use this command to drop a logical connection when you are dismantling a warm standby application.
- Before you can drop the logical connection, you must drop the connection to the standby database.

Permissions

drop logical connection requires "sa" permission.

See also

create connection, create logical connection, drop connection, switch active

drop partition

Description

Removes a disk partition from the Replication Server.

Syntax

drop partition *logical_name*

Parameters

logical_name
The name assigned to a partition created with create partition.

Examples

Drops the partition named P1 from the Replication Server:

```
drop partition P1
```

Usage

- Use drop partition to remove a disk partition. This command first marks the partition as "pending drop." Once it is marked, no new data is written on the partition.

  After all of the data stored on the partition has been successfully delivered, the partition is dropped.

  Note If not all the data stored on the partition is ready to drop, drop partition can create confusing behavior. For example, when a partition queue contains a segment that is filled only partially, the queue cannot drop until the segment is filled. Since the partition is designated "pending drop," the segment cannot fill, and the command fails to drop the partition.

- For a complete discussion of recovering from failed partitions, see the *Replication Server Administration Guide Volume 2*.

Permissions

drop partition requires "sa" permission.

See also

admin disk_space, alter partition, create partition

drop publication

Description

Drops a publication and all of its articles, and optionally drops the replication definitions for the articles.

Syntax

drop publication *pub_name*
with primary at *data_server.database*
[drop_repdef]

Parameters

pub_name
The name of the publication to drop.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

drop_repdef
An optional keyword that causes the replication definitions for the publication's articles to be dropped—if it is not used elsewhere.

Examples

Example 1 Drops the publication named pubs2_pub for the primary database TOKYO_DS.pubs2:

```
drop publication pubs2_pub
 with primary at TOKYO_DS.pubs2
```

Example 2 Drops the publication named pubs2_pub for the primary database TOKYO_DS.pubs2. This command also drops all the replication definitions for the publication's articles, for replication definitions that are not used elsewhere:

```
drop publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 drop_repdef
```

Usage

- Use drop publication to remove a publication. Execute drop publication at the Replication Server that manages the database where the primary data is stored.
- You can drop a publication if there are no subscriptions for the publication. Drop subscriptions first, as necessary.
- When you drop a publication, its articles are also dropped. Optionally, you can also drop all of the replication definitions for the publication's articles, if they are not part of any other article and have no subscriptions.

- A dropped publication is removed from a replicate site when define/create subscription or check publication is executed there for the publication.

Permissions	drop publication requires "create object" permission.
See also	check publication, create publication, drop article, drop function replication definition, drop replication definition, drop subscription

drop replication definition

Description

Drops a replication definition and its functions.

Syntax

drop replication definition *replication_definition*

Parameters

replication_definition
The name of the replication definition to drop.

Examples

Drops the replication definition named publishers_rep and any function strings that exist for it:

```
drop replication definition publishers_rep
```

Usage

- Use drop replication definition to remove a replication definition. Before a replication definition can be dropped, all subscriptions for it must be dropped.
- Execute drop replication definition at the primary Replication Server for the replication definition.
- If the dropped replication definition is the last replication definition for a primary table stored in an Adaptive Server, then, execute sp_setreplicate in the database after the replication definition is dropped. Set the table's replicate status to false to stop the Adaptive Server from logging special replication records for the table.
- If you use more than one version of Replication Server (for example, Replication Server version 11.5 and version 11.0.*x*) and create multiple replication definitions for the same primary table, the first replication definition created, which has the same primary and replicate table names, the same primary and replicate column names, and does not include table owner name, is marked and propagated to Replication Servers of version 11.0.*x* or earlier.

  When a replication definition that was propagated to a Replication Server of version 11.0.*x* or earlier is dropped, the oldest replication definition (if there is one) compatible with 11.0.*x* is propagated to 11.0.*x* or earlier sites. See create replication definition for more information about working with replication definitions in a mixed-version environment.
- Replication Server distributes information about the dropped replication definition to qualifying sites through the replication system. The changes do not appear immediately at all sites because of normal replication system lag time.

Permissions

drop replication definition requires "create object" permission.

See also — alter replication definition, check subscription, create replication definition, create subscription, define subscription, drop article, drop publication, drop subscription

drop route

Description

Closes the route to another Replication Server.

Syntax

drop route to *dest_replication_server* [with nowait]

Parameters

dest_replication_server
The name of the Replication Server whose route is to be dropped.

with nowait
Instructs Replication Server to close the route, even if it cannot communicate with the destination Replication Server. Use with nowait only as a last resort. This clause forces Replication Server to drop a route that has subscriptions or is used by an indirect route. Additional steps are usually required to remove the invalid references from the RSSDs of the affected Replication Servers.

Examples

Drops the route from the site where the command is entered to the SYDNEY_RS Replication Server:

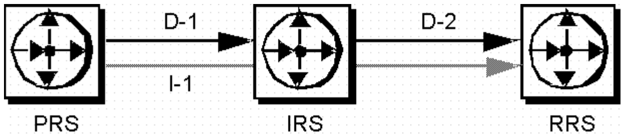
```
drop route to SYDNEY_RS
```

Usage

- drop route closes the route from the Replication Server where it is entered to the specified Replication Server.
- Before dropping a route, you must:
  - At the destination Replication Server, drop all subscriptions for primary data in databases managed by the source Replication Server.
  - Drop any indirect routes that use the route.

  For example, in Figure 3-4, route I-1 is an indirect route from the primary Replication Server (PRS) to the replicate Replication Server (RRS) via the intermediate Replication Server (IRS). It uses direct routes D-1 and D-2.

Figure 3-4: Example of direct and indirect routes



Before you can drop direct route D-2, you must drop all subscriptions at the replicate Replication Server for replication definitions at the primary or intermediate Replication Server, then drop indirect route I-1.

Warning! Use the with nowait clause only if you will never use the destination Replication Server again or if you must drop the route from the source Replication Server while the destination Replication Server is unavailable. Avoid the with nowait clause whenever possible so that the destination Replication Server can be updated correctly.

- After dropping a route using with nowait, you can use sysadmin purge_route_at_replicate at the (former) destination site to remove subscriptions and route information from the system tables at the destination.
- If the Replication Server from which the route is to be dropped is an intermediate site for another Replication Server, the route cannot be dropped. See the *Replication Server Administration Guide Volume 1* for more information.
- For Replication Servers with ERSSD, if the route being dropped is the last route originating from this source, then:
  - ERSSD Replication Agent is shut down
  - Log transfer is turned off from the ERSSD at the end of dropping the route

Permissions

drop route requires “sa” permission.

See also

alter route, create connection, create route, sysadmin purge_route_at_replicate

drop subscription

Description

Drops a subscription to a database replication definition, table replication definition, function replication definition, article, or publication.

Syntax

```
drop subscription sub_name
for {table_rep_def | function_rep_def |
{article article_name in pub_name |
   publication pub_name | database replication definition db_repdef
      with primary at data_server.database }
 with replicate at data_server.database
[without purge [with suspension
     [at active replicate only]] |
  [incrementally] with purge]
```

Parameters

sub_name
The name of the subscription to drop. If you are dropping a subscription for an article within a publication, specify the publication subscription name.

for *table_rep_def*
Specifies the name of the table replication definition the subscription is for.

for *function_rep_def*
Specifies the name of the function replication definition the subscription is for.

for article *article_name* in *pub_name*
Specifies the name of the article the subscription is for and the name of the publication that contains the article.

for publication *pub_name*
Specifies the name of the publication the subscription is for.

for database replication definition *db_repdef*
Specifies the name of the database replication definition the subscription is for.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database. Include this clause only for a subscription to a publication or a subscription to an article.

with replicate at data_server.database
Specifies the location of the replicate data. If the replicate database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

without purge

Instructs Replication Server to leave rows replicated by a subscription in the replicated copy.

A subscription to a function replication definition is always dropped without purging replicate data. For a subscription to a table replication definition or a publication, you must choose either without purge or with purge. For a subscription to a database replication definition, you must include without purge.

with suspension

Used with the without purge clause, suspends the DSI after the subscription is dropped so that you can manually delete subscription rows. If the database is part of a warm standby application, with suspension suspends the DSI threads for the active and the standby databases. Delete subscription rows from both databases.

with suspension at active replicate only

Used with the without purge clause, suspends the DSI after the subscription is dropped so that you can manually delete subscription rows. In a warm standby application, the standby DSI is not suspended. This allows Replication Server to replicate delete transactions from the active database to the standby database.

incrementally

Used with the with purge clause, specifies that deletes are made 10 rows at a time.

with purge

Used with a table replication definition, article, or publication, instructs Replication Server to remove rows (in the replicate table) that were replicated by a subscription.

A subscription to a function replication definition is always dropped without purging replicate data. For a subscription to a table replication definition or a publication, you must choose either without purge or with purge.

Examples

Example 1 Drops the authors_sub subscription for the authors_rep table replication definition. The replicate data is in the pubs2 database of the SYDNEY_DS data server. The rows replicated via the subscription are purged from the replicate table, where they are not part of another subscription:

```
drop subscription authors_sub
  for authors_rep
    with replicate at SYDNEY_DS.pubs2
    with purge
```

Example 2 Drops the titles_sub subscription for the titles_rep table replication definition. The replicate data is in the pubs2 database of the SYDNEY_DS data server. The rows replicated via the subscription remain in the replicate table:

```
drop subscription titles_sub
  for titles_rep
    with replicate at SYDNEY_DS.pubs2
    without purge
```

Example 3 Drops the myproc_sub subscription for the myproc_rep function replication definition. The replicate data is in the pubs2 database of the SYDNEY_DS data server. No subscription data is purged:

```
drop subscription myproc_sub
  for myproc_rep
    with replicate at SYDNEY_DS.pubs2
```

Example 4 Drops the subscription for the article titles_art that is part of the subscription pubs2_sub for the publication pubs2_pub. The primary data is in the pubs2 database of the TOKYO_DS data server and the replicate data is in the pubs2 database of the SYDNEY_DS data server. The rows that were replicated via the subscription remain in the affected replicate tables. After dropping the article subscription you can drop the article:

```
drop subscription pubs2_sub
  for article titles_art in pubs2_pub
    with primary at TOKYO_DS.pubs2
    with replicate at SYDNEY_DS.pubs2
    without purge
```

Example 5 Drops the subscription named pubs2_sub for the pubs2_pub publication, where the primary data is in the pubs2 database of the TOKYO_DS data server and the replicate data is in the pubs2 database of the SYDNEY_DS data server. The rows that were replicated via the subscription are purged from the affected replicate tables, where they are not part of another subscription:

```
drop subscription pubs2_sub
  for publication pubs2_pub
    with primary at TOKYO_DS.pubs2
    with replicate at SYDNEY_DS.pubs2
    with purge
```

Example 6 Deletes a database subscription named pubs2_sub. The without purge option ensures that Replication Server does not remove rows added by the subscription to the replicate:

```
drop subscription pubs2_sub
  for database replication definition pubs2_rep
    with primary at NEWYORK_DS.pubs2
    with replicate at TOKYO_DS.pubs2
    without purge
```

Usage

- When you drop a subscription, Replication Server stops replicating the data specified by the subscription.
- Execute drop subscription at the Replication Server where you created the subscription.
- You cannot drop a table replication definition, function replication definition, article, or publication until you have dropped all subscriptions for the object.

The *without purge* clause

- Use without purge to drop a subscription to a table or database replication definition or to a publication. Replicated rows remain in the replicate tables.
- When you drop a subscription to a table replication definition or publication, you *must* specify either without purge or with purge.
- When you drop a subscription to a function replication definition, it is *always* dropped "without purge"—you do not need to specify without purge.
- When you drop a publication subscription "without purge," all of its article subscriptions are dropped together.

The *with purge* clause

- Use the with purge clause to delete the rows (in the replicate table) that were replicated by the subscription. All subscription rows are purged unless they belong to another subscription at the replicate site.
- When you use with purge, Replication Server selects, from the replicate database, the set of rows that could be deleted. It then evaluates the selected rows against other subscriptions and determines whether to delete the row. The maintenance user for the replicate database must have select permission on the table.
- Deletes using with purge occur in a single transaction performed by an rs_select_with_lock function string in the replicate database.

- Deletes using with purge and incrementally occur 10 rows at a time. This operation is performed by an rs_select function string in the replicate database.
- When you drop a publication subscription "with purge," its article subscriptions are dropped one at a time in the reverse order that the articles were added to the publication.

Permissions

drop subscription requires "create object" permission at the replicate site and "primary subscribe" permission at the primary Replication Server.

drop subscription ... with purge also requires that the maintenance user have select permission for the replicate table.

See also

check subscription, create subscription, define subscription, drop article, drop function replication definition,drop publication, drop replication definition, resume connection, rs_select, rs_select_with_lock

drop user

Description

Drops a Replication Server user login name.

Syntax

drop user *user*

Parameters

user
The user login name to be dropped.

Examples

Removes the login name "louise" from the Replication Server:

```
drop user louise
```

Usage

- Use drop user to remove a Replication Server login name.
- Execute this command on the Replication Server where the login name was created.

Permissions

drop user requires "sa" permission.

See also

alter user, create user

grant

Description

Assigns permissions to users.

Syntax

```
grant {sa | create object | primary subscribe |
 connect source}
 to user
```

Parameters

sa
Users with "sa" permission can execute any RCL command.

create object
Allows the recipient to create, alter, and drop Replication Server objects, such as replication definitions, subscriptions, and function strings.

primary subscribe
Allows recipient to create subscriptions for a replicated table whose primary data is managed by the current Replication Server.

connect source
This permission is granted to RepAgents and other Replication Servers to log in to the Replication Server.

user
The login name of a user who is to receive the permission.

Examples

Example 1 Allows the user "thom" to execute any Replication Server command:

```
grant sa to thom
```

Example 2 Allows the user "louise" to create subscriptions:

```
grant primary subscribe to louise
```

Usage

- The "sa" permission cannot be revoked from the "sa" user.
- The "connect source" permission is needed by the RSI or RepAgent. Refer to the Replication Server installation and configuration guides for your platform for more information.
- For each RCL command described in this manual, the minimum permission required to execute the command is shown. For a list of minimum permissions for all commands, see the *Replication Server Administration Guide Volume 1.*

Permissions

grant requires "sa" permission.

See also

revoke

ignore loss

Description

Allows Replication Server to accept messages after it detects a loss.

Syntax

```
ignore loss
 from data_server.database
[to {data_server.database | replication_server}]
```

Parameters

from *data_server*.*database*
Specifies the primary data server and database whose message loss is to be ignored.

to *data_server*.*database*
Specifies the destination data server and database for the lost messages.

to *replication_server*
Specifies the destination Replication Server for the lost messages.

Usage

- Replication Server detects loss when it rebuilds queues or replays transaction logs in recovery mode.
- A Replication Server detects message losses on connections to the replicate databases it manages.
- For warm standby databases, use the logical connection name for *data_server*.*database*, except for losses that Replication Server detects between the active database and the standby database. To ignore these losses, use the physical *data_server*.*database* name.
- If direct routes exist, the destination Replication Server detects message losses from the source Replication Server. Look in both Replication Server log files to determine whether losses were detected.
- When a Replication Server detects losses, it accepts no messages on the connection until ignore loss is executed.
- After ignore loss is executed, a few updates may be necessary before messages begin to flow again.
- After ignore loss is executed, procedures are required to bring replicated data up to date.

See the *Replication Server Administration Guide Volume 2* for detailed recovery instructions.

Permissions

ignore loss requires "sa" permission.

See also

allow connections, configure route, rebuild queues, set log recovery

move primary

Description

Changes the primary Replication Server for an error class or a function-string class.

Syntax

move primary
 of {error class | function string class} *class_name*
to *replication_server*

Parameters

error class
Specifies that the primary Replication Server for an error class is to be changed.

function string class
Specifies that the primary Replication Server for a function-string class is to be changed.

class_name
The name of the error class or function-string class whose primary Replication Server is to be changed.

replication_server
Specifies the new primary Replication Server for the error class or function-string class. It is the name of the Replication Server where the command is executed, since move primary must be executed at the new primary Replication Server.

Examples

Example 1 Changes the primary Replication Server for the pubs2_db_err_class error class to the SYDNEY_RS Replication Server. The command is entered at SYDNEY_RS:

```
move primary
 of error class pubs2_db_err_class
 to SYDNEY_RS
```

Example 2 Changes the primary Replication Server for the sqlserver2_function_class function-string class to the SYDNEY_RS Replication Server. The command is entered at SYDNEY_RS:

```
move primary
 or function string class sqlserver2_function_class
 to SYDNEY_RS
```

Usage

- If you have changed the routing configuration, use move primary to ensure that error responses and function strings are distributed, via the new routes, to the Replication Servers where they are needed.

- move primary must be executed at the new primary Replication Server.
- move primary can be used to change the primary Replication Server from A to B only if routes exist from A to B and from B to A.
- There is no primary site for the system-provided rs_sqlserver_function_class until you assign one. rs_default_function_class and rs_db2_function_class are system-provided, cannot be modified, and have no primary site.
- The primary site for a derived function-string class is the site of its parent class, unless the parent class is rs_default_function_class or rs_db2_function_class. In that case, the primary site of the derived class is the site where it was created.
- If you use rs_sqlserver_function_class, you must specify a primary site before you can modify a default function-string. To specify a primary site for the function-string class, execute create function string class rs_sqlserver_function_class at the primary site. Then use the move primary command to change the primary site for the class.
- There is no primary site for the default error class, rs_sqlserver_error_class, until you assign one. You must specify a primary site before you use assign action to change default error actions. To specify a primary site, execute create error class rs_sqlserver_error_class at the primary site. Then you can use move primary to change the primary site.

Permissions

move primary requires "sa" permission.

See also

alter route, assign action, create error class, create function string class

rebuild queues

Description

Rebuilds Replication Server stable queues.

Syntax

```
rebuild queues
```

Usage

- Rebuild stable queues to recover from a failed or missing partition.

Warning! Use this command only as described in the *Replication Server Administration Guide Volume 2*. rebuild queues *deletes* messages from the replication system and may make it more difficult to correct other problems.

- Drop damaged partitions and replace them, if necessary, before you rebuild queues. A dropped partition may not actually be removed from the system until rebuild queues is executed.
- rebuild queues disconnects all other Replication Servers from the Replication Server where it is executed. Connection attempts are refused until the queues are rebuilt.
- rebuild queues clears all of the Replication Server's stable queues, and "gives up" any damaged partitions in use.
- If you start Replication Server in stand-alone mode (using the -M command line flag) and then execute rebuild queues, Replication Server goes into recovery mode.
- While restoring messages to the rebuilt stable queues, Replication Server determines whether the data cleared from the queues was recovered or lost. Look for error messages in the log file of the Replication Server with the rebuilt queues and in the log files of Replication Servers that have direct routes from it. Loss detection may not complete immediately; it is necessary for new data to flow from each primary database or upstream site.
- If loss is detected, you may need to re-create subscriptions or recover data from offline dumps.
- If a subscription is materializing when you use rebuild queues, drop and re-create it. Even if the materialization appears to have completed successfully, some data may have been lost.
- After queues are rebuilt, the Replication Server attempts to restore lost messages by requesting backlogged messages from Replication Servers that have routes to the current Replication Server.

- You cannot rebuild queues for specific database connections or routes.

For help with recovery procedures, see the *Replication Server Administration Guide Volume 2*.

Permissions

rebuild queues requires "sa" permission.

See also

add partition, alter partition, configure connection, create partition, drop partition, ignore loss, resume log transfer, set log recovery

resume connection

Description

Resumes a suspended connection.

Syntax

```
resume connection
 to data_server.database
[skip transaction | execute transaction]
```

Parameters

data_server
The name of the data server that holds the database whose connection is to be resumed.

database
The name of the database whose connection is to be resumed.

skip transaction
instructs Replication Server to resume execution with the second transaction in the connection's queue. The first transaction is written to the database exceptions log.

execute transaction
overrides the Replication Server restriction against the application of system transactions after a DSI startup if the system transaction is the first transaction in the DSI queue.

Examples

Resumes the connection to the pubs2 database in the SYDNEY_DS data server:

```
resume connection to SYDNEY_DS.pubs2
```

Usage

- Resuming a connection allows replication activities for the suspended database to begin again.
- Suspend connections so you can alter them with alter connection or perform maintenance on the suspended database. Connections are also suspended during subscription materialization or dematerialization.
- Replication Server can suspend a database connection because of an error.
- resume connection is also used to resume a connection suspended because of an error.
- If you determine that the system transaction was executed, use the skip transaction clause.
- Use the execute transaction clause *only* if a system transaction has failed to execute and you have corrected the problem that prevented its execution. A system transaction has no enclosing begin tran/commit tran pair. If Replication Server is restarted with a system transaction as the first transaction, you see this message:

```
E. 1998/02/16 14:43:49. ERROR #5152 DSI (206 hookip01.rdb1) - dsisched.c
(2196)
 There is a system transaction whose state is not known. DSI will be shut
down.
```

Determine whether the database has executed this transaction and use skip transaction or execute transaction as appropriate.

Permissions

resume connection requires “sa” permission.

See also

activate subscription, alter connection, assign action, create connection, drop connection, drop subscription, suspend connection

resume distributor

Description

Resumes a suspended Distributor thread for a connection to a database.s

Syntax

resume distributor *data_server*.*database* [skip transaction]

Parameters

data_server
The data server name. If the database is part of a warm standby application, *data_server* is the logical data server name.

database
The database name. If the database is part of a warm standby application, *database* is the logical database name.

skip transaction
Instructs Replication Server to resume execution with the second transaction in the connection's queue. The first transaction is written to the database exceptions log.

Examples

Resumes the Distributor thread for the logical data server LDS and the pubs2 database:

```
resume distributor LDS.pubs2
```

Usage

- Use resume distributor to resume a Distributor thread suspended using suspend distributor or suspended by Replication Server.
- Use skip transaction to resume connection when distributor is down due to:
  - message in inbound queue is longer than 16,000 bytes and site version has not been upgraded to RS 12.5 and up, or
  - downstream Replication Server cannot accept new feature commands, for example, bigint.

Permissions

resume distributor requires "sa" permission.

See also

suspend distributor

resume log transfer

Description

Allows the RepAgent to connect to the Replication Server.

Syntax

resume log transfer
 from {*data_server*.*database* | all}

Parameters

data_server
the name of the data server with the database whose RepAgent is to be connected to the Replication Server.

database
the database whose RepAgent is to connect to the Replication Server.

all
permits RepAgents for all databases managed by the Replication Server to connect.

Examples

Example 1 The Replication Server will accept connections from any RepAgent:

```
resume log transfer from all
```

Example 2 The Replication Server will accept a connection from a RepAgent for the pubs2 database in the SYDNEY_DS data server:

```
resume log transfer from SYDNEY_DS.pubs2
```

Usage

- When you quiesce a Replication Server or the replication system, use suspend log transfer to cause Replication Server to refuse RepAgent connections.
- resume log transfer allows the RepAgent threads to connect to a Replication Server upon which suspend log transfer has been executed.
- Normally, the RepAgent retries its connection to Replication Server following a suspend log transfer until resume log transfer allows it to reconnect. However, if the RepAgent is down for any reason, resume log transfer does not restart it.
- After resuming log transfer from ERSSD, the recovery daemon will automatically restart the ERSSD RepAgent when it wakes up.

Permissions

resume log transfer requires “sa” permission.

See also

admin quiesce_check, admin quiesce_force_rsi, resume connection,

resume queue

Description

Restarts a stable queue stopped after being passed a message larger than 16K bytes. Applicable only when the Replication Server version is 12.5 or later and the site version has not been similarly upgraded.

Syntax

resume queue, *q_number*, *q_type* [, skip transaction with large message]

Parameters

q_number
The queue number of the stable queue.

q_type
The queue type of the stable queue. Values are “0” for outbound queues, “1” for inbound queues.

skip transaction with large message
Specifies that the SQM should skip the first large message encountered after restarting.

Examples

Specifies that outbound queue #2 skips the first large message it is passed by the RepAgent:

```
resume queue, 2, 0, skip transaction with large message
```

Usage

- This command is applicable only when the Replication Server is version 12.5 or later and the site version is not upgraded.
- resume queue does not skip any messages if the site version is 12.5 or later.

Permissions

alter queue requires “sa” permission.

See also

alter queue

resume route

Description

Resumes a suspended route.

Syntax

```
resume route
 to dest_replication_server [skip transaction with large message]
```

Parameters

dest_replication_server
The name of the destination Replication Server; that is, the suspended route you want to resume.

skip transaction with large message
Ignore first transaction encountered with a message greater than 16,000 bytes.

Examples

Resumes the route to the SYDNEY_RS Replication Server:

```
resume route to SYDNEY_RS
```

Usage

- Resuming a route allows Replication Server to begin sending queued messages to the remote Replication Server again.
- resume route can also be used to resume a route suspended because of an error.
- skip transaction with large message is applicable only to direct routes where the site version at the replicate site is 12.1 or earlier.

Permissions

resume route requires “sa” permission.

See also

alter route, create route, drop route, suspend route

revoke

Description	Revokes permissions from users.

Syntax

```
revoke {sa | connect source | create object |
 primary subscribe}
 from user
```

Parameters

sa
: Denies permission to execute commands that require “sa” permission.

connect source
: Denies permission to execute RCL commands used by RepAgents or other Replication Servers.

create object
: Denies permission to create, alter, and drop Replication Server objects such as replication definitions, subscriptions, and function strings.

primary subscribe
: Denies permission to create subscriptions for a replicated table if the primary data is managed by the current Replication Server.

user
: The login name of the user whose permission is to be revoked.

Examples

Example 1 Prevents user “thom” from executing commands that create or modify Replication Server objects:

```
revoke create object from thom
```

Example 2 Prevents user “louise” from creating subscriptions for primary data managed by this Replication Server, unless she has “create object” or “sa” permission at the primary Replication Server:

```
revoke primary subscribe from louise
```

Usage

- revoke requires “sa” permission.
- The “sa” permission cannot be revoked from the “sa” user login name.

Permissions

revoke requires “administrator” permission.

See also

create replication definition, check subscription, create user, grant

set autocorrection

Description

Prevents failures that would otherwise be caused by missing or duplicate rows in a replicated table.

Syntax

set autocorrection {on | off}
 for *replication_definition*
with replicate at *data_server*.*database*

Parameters

on
Enables autocorrection for the specified replication definition.

off
Disables autocorrection for the specified replication definition.

replication_definition
The name of the replication definition whose autocorrection status you are changing.

data_server
The name of the data server with the replicate database for which you are changing the autocorrection status. If the replicate database is part of a warm standby application, *data_server* is the logical data server name.

database
The name of the replicate database where you are changing the autocorrection status. If the replicate database is part of a warm standby application, *database* is the logical database name.

Examples

Enables autocorrection for the publishers_rep replication definition in the pubs2 database at the SYDNEY_DS data server:

```
set autocorrection on
 for publishers_rep
 with replicate at SYDNEY_DS.pubs2
```

Usage

- Use set autocorrection to prevent duplicate key errors that might occur during non-atomic materialization.
- Autocorrection should be enabled *only* for replication definitions whose subscriptions use non-atomic materialization (create subscription specified without holdlock). After materialization is complete and the subscription is VALID, disable autocorrection to improve performance.
- Autocorrection is off, by default, for a replication definition.

How autocorrection works

- set autocorrection determines how Replication Server processes inserts and updates to replicated tables. When autocorrection is on, Replication Server converts each update or insert operation into a delete followed by an insert.

  For example, if a row inserted into the primary version of a table already exists in a replicated copy and autocorrection is off, the operation results in an error. When autocorrection is on, Replication Server converts the insert to a delete followed by an insert so that the insert cannot fail because of an existing row.

  If the primary key has changed in a row that is to be replicated, Replication Server deletes two rows in the replicated table before it inserts the row. It deletes the row in which the primary key matches the before image and the row in which the primary key matches the after image.

- When autocorrection is on, an insert or update at a primary database may cause delete and insert triggers to fire at the replicate database. The delete trigger fires only if the row inserted or updated at the primary database was already present at the replicate database.

- Replication Server creates entries for replication definitions with autocorrection enabled in the rs_repobjs system table.

Autocorrection and replicated stored procedures

- Replication Server does not perform autocorrection for rows updated at replicate databases as the result of using replicated stored procedures that modify primary data. See the *Replication Server Administration Guide Volume 1* for more information about replicating stored procedures.

Note If you use replicated stored procedures to modify primary data, be sure to write stored procedures at the replicate Replication Server to correct for the failed updates and inserts that can occur during non-atomic materialization. Stored procedures at the replicate Replication Server should simulate autocorrection, treating update and insert operations as combined delete-insert operations. Alternatively, stored procedures can correct failed updates and inserts after they are detected.

Autocorrection and *replicate minimal columns*

- If a replication definition uses replicate minimal columns, you cannot set autocorrection on. If you set autocorrection on before specifying minimal columns (for example, using alter replication definition), autocorrection is not performed. Replication Server logs informational messages for any update operations.

Autocorrection and *text*, *unitext*, or *image* datatypes

- If a replication definition has a text, unitext, or image column in the replicate_if_changed column list, an attempt to enable autocorrection for the replication definition causes an error. Autocorrection requires that all text, unitext, and image columns appear in the always_replicate list for the replication definition.

Permissions

set autocorrection requires "create object" permission.

See also

alter replication definition, create replication definition, create subscription

set log recovery

Description	Specifies databases whose logs are to be recovered from offline dumps.

Syntax

```
set log recovery
 for data_server.database
```

Parameters

data_server
The data server with the database to be recovered.

database
The database to be recovered.

Usage

- Execute set log recovery after restarting Replication Server in stand-alone mode.
- Execute allow connections after set log recovery to enter recovery mode. Replication Server accepts connections only from RepAgents started in recovery mode for databases named in set log recovery. This ensures that old log records are replayed before new log records are accepted.

See the *Replication Server Administration Guide Volume 2* for detailed recovery procedures.

Permissions

set log recovery requires "sa" permission.

See also

allow connections, ignore loss, rebuild queues

set proxy

Description

Switches to another user.

Syntax

set proxy [to] [*user_name* [verify password *passwd*]]

Parameters

user_name
A valid Replication Server login name.

verify password
Verifies the password of a Replication Server user.

passwd
the password of a valid Replication Server user.

Usage

- set proxy *user_name* switches to a new user with all the permissions of the new user and none of the permissions of the original user.
- The new user can always switch back to the original user, whether or not the new user has “sa” permission, by entering set proxy without a user name.
- set proxy *user_name* verify password *passwd* allows a user without sa permission to switch to another user—if the correct password for *user_name* is entered.

Permissions

set proxy *user_name* requires “sa” permission. Any user can execute set proxy and set proxy *user_name* verify password *passwd*.

See also

alter connection, alter route, configure replication server, create connection, create route

shutdown

Description	Shuts down a Replication Server.
Syntax	shutdown
Examples	Instructs the Replication Server to shut down:

```
shutdown
```

Usage	Use the shutdown command to shut down a Replication Server. This command instructs Replication Server to refuse additional connections, terminate processes, and exit.
Permissions	shutdown requires “sa” permission.

suspend connection

Description

Suspends a connection to a database.

Syntax

```
suspend connection
 to data_server.database
[with nowait]
```

Parameters

data_server
The name of the data server with the database whose connection is to be suspended.

database
The name of the database whose connection is to be suspended.

with nowait
Suspends the connection immediately.

Examples

Suspends the connection to the pubs2 database in the SYDNEY_DS data server:

```
suspend connection to SYDNEY_DS.pubs2
```

Usage

- Suspending a connection temporarily halts replication activities for the database.
- Connections are suspended so they can be altered with alter connection or so that maintenance can be performed. You can also use suspend connection to control when replicate databases are updated.
- While a connection is suspended, Replication Server holds transactions for the database in stable queues.
- If suspend connection is executed without the with nowait clause, Replication Server attempts to complete any transaction that is in progress. However, the connection to the data server may be suspended before the transaction is completed.
- To reactivate the connection, use resume connection.

Permissions

suspend connection requires “sa” permission.

See also

alter connection, create connection, drop connection, resume connection

suspend distributor

Description

Suspends the Distributor thread for a connection to a primary database.

Syntax

suspend distributor *data_server.database*

Parameters

data_server
The data server name. If the database is part of a warm standby application, *data_server* is the logical data server name.

database
The database name. If the database is part of a warm standby application, *database* is the logical database name.

Examples

Suspends the Distributor thread for the pubs2 database in the LDS data server:

```
suspend distributor LDS.pubs2
```

Usage

- Use suspend distributor to suspend a Distributor thread for a logical or physical connection to a primary database.
- To resume the Distributor thread, use resume distributor.
- The distributor thread reads incoming primary database transactions and forwards them to subscribers. Turn off the distributor to enhance performance in a warm-standby-only environment that has only a standby database and no subscribers.

Permissions

suspend distributor requires “sa” permission.

See also

resume distributor

suspend log transfer

Description

Disconnects a RepAgent from a Replication Server and prevents a RepAgent from connecting.

Syntax

suspend log transfer
 from {*data_server*.*database* | all}

Parameters

data_server
The data server with the database whose RepAgent is to be suspended.

database
The database whose RepAgent is to be suspended or whose connections are to be disallowed.

all
Instructs Replication Server to suspend all RepAgents and to disallow future connections for all RepAgents.

Examples

Example 1 Disconnects the RepAgent for the pubs2 database and does not permit it to reconnect:

```
suspend log transfer from TOKYO_DS.pubs2
```

Example 2 Disconnects all connected RepAgents and does not permit any RepAgent to reconnect to the Replication Server:

```
suspend log transfer from all
```

Usage

- Use suspend log transfer to disconnect a RepAgent. This is the first step in quiescing the replication system. suspend log transfer does not shut down the RepAgent.
- To test whether the system is quiesced after suspending a RepAgent, use admin quiesce_check.
- To allow RepAgents to connect to the Replication Server, execute resume log transfer.

Permissions

suspend log transfer requires “sa” permission.

See also

admin quiesce_check, admin quiesce_force_rsi, resume log transfer

suspend route

Description	Suspends a route to another Replication Server.
Syntax	suspend route to *dest_replication_server*
Parameters	*dest_replication_server* The name of the destination Replication Server, the route to which is to be suspended.
Examples	Suspends the route to the SYDNEY_RS Replication Server: `suspend route to SYDNEY_RS`
Usage	• Use suspend route to suspend a route to another Replication Server. This command lets you manage network use by controlling when messages are sent from one Replication Server to another. • While a route is suspended, Replication Server holds messages for the destination Replication Server in a stable queue. • You can suspend only direct routes. • To reactivate a suspended route, use resume route.
Permissions	suspend route requires “sa” permission.
See also	alter route, resume connection, resume route, suspend connection

switch active

Description

Changes the active database in a warm standby application.

Syntax

switch active
 for *logical_ds*.*logical_db*
to *data_server*.*database*
[with suspension]

Parameters

logical_ds
The logical data server name for the logical connection.

logical_db
The logical database name for the logical connection.

data_server
The data server name of the new active database for the logical connection.

database
The database name of the new active database for the logical connection.

with suspension
Suspends the DSI connection to the new active database after the switch is complete.

Examples

This command starts the switch active process:

```
switch active for LDS.pubs2 to OSAKA.pubs2
```

```
Switch of the active for this logical database is in
progress.
```

Usage

- switch active is a part of the procedure for switching to the standby database in a warm standby application. See the *Replication Server Administration Guide Volume 2* for the complete procedure.
- switch active returns immediately, but the switch is not complete until admin logical_status displays “None” in the State of Operation in Progress.
- Use admin logical_status to monitor the status of the switch active process.
- If you use the with suspension option, you must manually resume the DSI connection to the new active database after the switch is complete.
- After entering switch active, you can attempt to cancel it using abort switch.

Permissions

switch active requires “sa” permission.

See also

abort switch, admin logical_status, create logical connection, wait for switch

sysadmin apply_truncate_table

Description

Turns on or off the "subscribe to truncate table" option for all existing subscriptions to a particular table, enabling or disabling replication of truncate table.

Syntax

sysadmin apply_truncate_table, *data_server*,
 database, {*table_owner* | '' | ""}, *table_name*
{'on'| 'off'}You

Parameters

data_server
The name of the replicate data server.

database
The name of the replicate database managed by the data server.

table_owner
Identifies the owner of the replicate table. If owner is not specified, Replication Server sets owner to "dbo."

table_name
Identifies the replicate table for which you want to turn on or off the "subscribe to truncate table" option for existing subscriptions.

on
Turns on the "subscribe to truncate table" option for existing subscriptions.

off
Turns off the "subscribe to truncate table" option for existing subscriptions.

Examples

Turns on "subscribe to truncate table" for all subscriptions to the publishers table owned by emily in the pubs2 database:

```
sysadmin apply_truncate_table, SYDNEY_DS,
 pubs2, emily, publishers, 'on'
```

Usage

- Use sysadmin apply_truncate_table with Adaptive Server version 11.5 or later databases.
- If you did not specify a replicate table owner in the replication definition, enter '' (two single-quote characters) or "" (two double-quote characters) for the table owner name.
- Subscriptions for a particular table for a particular database must all support or not support replication of truncate table. If, for example, sysadmin apply_truncate_table is off, you cannot create new subscriptions that include the "subscribe to truncate table" option unless you turn sysadmin apply_truncate_table on for all subscriptions for that table.

See create subscription or define subscription for more information about setting the "subscribe to truncate table" option for new subscriptions.

- Replication Server executes truncate table at the replicate database as the maintenance user. Among the permissions granted to maintenance user is "replication_role." If you revoke maintenance user's "replication_role," you will be unable to replicate truncate table unless
  - The maintenance user has been granted "sa_role,"
  - The maintenance user owns the table, or
  - The maintenance user is aliased as the Database Owner.
- It is not necessary for warm standby databases to subscribe to truncate table; execution of the truncate table command is automatically replicated to standby databases. Turn on replication of truncate table for standby databases with the alter logical connection command.

Permissions

sysadmin apply_truncate_table requires "sa" permission.

See also

create subscription, define subscription

sysadmin dropdb

Description

Drops a database from the ID Server.

Syntax

sysadmin dropdb, *data_server*, *database*

Parameters

data_server
The name of the data server.

database
The name of the database you want to drop.

Examples

Drops the pubs2 database in the SYDNEY_DS data server from the ID Server:

```
sysadmin dropdb, SYDNEY_DS, pubs2
```

Usage

- Use sysadmin dropdb to drop a database from the ID Server. This command must be executed at an ID Server.
- Use sysadmin dropdb only when the ID Server system tables contain information about a database that does not exist in the system. This should happen only after a system failure.

  For example, if a database is dropped with drop connection, a network failure might prevent the ID Server from being notified so that it can remove the database from its tables. If you attempt to add the same data server and database to the system later, the request will fail because the database and its data server are already registered in the ID Server system tables.
- If you reinstall a Replication Server, use sysadmin dropdb to remove the ID Server information for each database the Replication Server managed, including its RSSD. Otherwise, errors occur when you reinstall Replication Server.
- If you enter invalid arguments with this command, you are not notified.

Warning! Never use sysadmin dropdb on any databases that have active connections.

Permissions

sysadmin dropdb requires “sa” permission.

See also

sysadmin dropldb

sysadmin dropldb

Description
Drops a logical database from the ID Server.

Syntax
sysadmin dropldb, *data_server*, *database*

Parameters
data_server
The name of the logical data server.

database
The name of the logical database you want to drop.

Examples
Drops the pubs2 logical database in the LDS logical data server from the ID Server:

```
sysadmin dropldb, LDS, pubs2
```

Usage

- Use sysadmin dropldb to drop a logical database from the ID Server. This command must be executed at an ID Server.
- Use sysadmin dropldb only when the ID Server system tables contain information about a logical database that does not exist in the system. This should happen only after a system failure.

  For example, if a logical database is dropped with drop logical connection, a network failure might prevent the ID Server from being notified so that it can remove the logical database from its tables. If you attempt to add the same logical data server and logical database to the system later, the request fails because the logical database and its logical data server are already registered in the ID Server system tables.
- If you reinstall a Replication Server, first use sysadmin dropldb to remove the ID Server information for each logical database the Replication Server managed. Otherwise errors occur when you reinstall Replication Server.
- If you enter invalid arguments with this command, you are not notified.

Warning! Never use sysadmin dropldb on any logical databases that have active connections.

Permissions
sysadmin dropldb requires “sa” permission.

See also
sysadmin dropdb

sysadmin drop_queue

Description

Deletes a stable queue. Use this command to drop a failed materialization queue.

Syntax

sysadmin drop_queue, *q_number*, *q_type*

Parameters

q_number
The site ID for the Replication Server or database that is the source or destination for the queue.

q_type
The queue type.

Usage

- Use sysadmin drop_queue to stop and delete a materialization queue that remains after a subscription experiences an unrecoverable error and must be manually cleaned up.

  Warning! Use sysadmin drop_queue only to drop a failed materialization queue.

- Use admin who to find the *q_number* and *q_type* for a queue. The values appear in the command's SQM thread output.

Permissions

sysadmin drop_queue requires "sa" permission.

See also

rebuild queues, sysadmin purge_route_at_replicate

sysadmin dropprs

Description

Drops a Replication Server from the ID Server.

Syntax

sysadmin dropprs, *replication_server*

Parameters

replication_server
The name of the Replication Server you want to drop.

Examples

Drops the SYDNEY_RS Replication Server from the ID Server:

```
sysadmin droprs, SYDNEY_RS
```

Usage

- Use sysadmin droprs to drop a Replication Server from the ID Server. This command can be executed only at an ID Server.
- You can use sysadmin droprs when the ID Server contains information about a Replication Server that does not exist in the replication system. Such a scenario is usually a result of a system failure. For example, if a Replication Server installation fails, the ID Server system tables may contain entries for the Replication Server, preventing subsequent attempts to install the Replication Server.
- You are not notified when you enter an invalid argument.

Warning! Use sysadmin droprs with caution when removing an active Replication Server. For the correct procedure on removing an active Replication Server, see the *Replication Server Administration Guide Volume 1*.

Permissions

sysadmin droprs requires “sa” permission.

sysadmin dump_file

Description

Specifies an alternative log file name for use when dumping a Replication Server stable queue.

Syntax

sysadmin dump_file [, *file_name*]

Parameters

file_name
The name of the new log file that stable queue dumps are to be written to.

Examples

Specifies pubs2.log as the file for logging stable queue output:

```
sysadmin dump_file, 'pubs2.log'
```

Usage

- Use sysadmin dump_file to specify a log file name before you use sysadmin dump_queue to dump the log to a file.
- To reset the current dump file to the default, execute sysadmin dump_file without specifying a file name.
- If a file name is specified, the current dump file is closed and a new file is opened. The new file uses the specified file name.
- The default dump file is the Replication Server log. Use admin log_name to display the path to this file.
- If you enter a log file name containing characters other than letters and numerals, enclose it in quotes.

Permissions

sysadmin dump_file requires “sa” permission.

See also

admin log_name, sysadmin dump_queue, sysadmin sqt_dump_queue

sysadmin dump_queue

Description

Dumps the contents of a Replication Server stable queue.

Syntax

sysadmin dump_queue, *q_number*, *q_type*,
seg, *blk*, *cnt* [, RSSD | client]

Parameters

q_number, *q_type*
Identifies the stable queue to dump. Find these values using admin who or admin who, sqm.

seg
Identifies the starting segment.

blk
Identifies the 16K block in the segment where the dump is to begin. Block numbering starts at 1 and ends at 64.

sysadmin dump_queue recognizes four special settings for *seg* and *blk*:

- Setting *seg* to -1 starts with the first active segment in the queue.
- Setting *seg* to -2 starts with the first segment in the queue, including any inactive segments retained by setting a save interval.
- Setting *seg* to -1 and *blk* to -1 starts with the first undeleted block in the queue.
- Setting *seg* to -1 and *blk* to -2 starts with the first unread block in the queue.

cnt
Specifies the number of blocks to dump. This number can span multiple segments. If *cnt* is set to -1, the end of the current segment is the last block dumped. If it is set to -2, the end of the queue is the last block dumped.

RSSD
Optional flag that forces the output to system tables in the RSSD.

client
Optional flag that forces the output to the client that is issuing this command.

Examples

Example 1 Acting on queue 103:1, dumps blocks 15–64 of segment 0 and blocks 1–15 of segment 1 into the Replication Server log:

```
sysadmin dump_queue, 103, 1, 0, 15, 65
```

Example 2 Dumps all of queue 103:1 into the Replication Server log:

```
sysadmin dump_queue, 103, 1, -1, 1, -2
```

Example 3 Dumps all of queue 103:1 into the RSSD:

```
sysadmin dump_queue, 103, 1, -1, 1, -2, RSSD
```

Example 4 Dumps all of queue 103:1 to the client:

```
sysadmin dump_queue, 103, 1, -1, 1, -2, client
```

Example 5 This series of commands dumps all of queue 103:1 into a file named SYDNEY_RS.log. The last sysadmin dump_file command closes the SYDNEY_RS.log file. Any subsequent dumps are directed to the Replication Server log:

```
sysadmin dump_file, SYDNEY_RS.log
 sysadmin dump_queue, 103, 1, -1, 1, -2
 sysadmin dump_file
```

Usage

- Use sysadmin dump_queue to dump the contents of a Replication Server stable queue.
- Output from sysadmin dump_queue may go to one of the following:
  - Replication Server log
  - Alternate log file
  - RSSD
  - Client issuing the command

  To dump queues into the RSSD or client, the last argument of sysadmin dump_queue must be RSSD or client.

  If the RSSD or client option is not specified, output goes into the Replication Server log.

  If an alternative log file for dumping queues is specified using the sysadmin dump_file command, the output goes into the alternative dump file.
- Specify the maximum command length used by this command by setting the queue_dump_buffer_size configuration parameter.

Dumping to the RSSD

- If the RSSD option is used, the dump is written into two system tables in the RSSD, rs_queuemsg and rs_queuemsgtxt.

  If the queue is dumped into the RSSD, the system tables are first cleared of the segments with the same *q_number*, *q_type*, *seg*, and *blk* as the blocks being dumped.

For information about the contents of the rs_queuemsg system table, see Chapter 8, "Replication Server System Tables."

The rs_queuemsgtxt system table holds the text of commands dumped from the stable queue. If the text of a command exceeds 255 characters, it is stored in multiple rows numbered with the q_seq column.

Dumping to the client

- If the client option is used, the dump is written to the client issuing the command, such as isql or Replication Server Manager.

Permissions

sysadmin dump_queue requires "sa" permission.

See also

admin who, rs_queuemsg, rs_queuemsgtxt, sysadmin dump_file

sysadmin dump_thread_stack

Description

Dumps Replication Server stacks.

Syntax

sysadmin dump_thread_stack [, *module_name*]

Parameters

module_name

The type of Replication Server thread. The valid module names are the same as the values under the name column displayed by the admin who command.

Examples

Dumps the RSI queue stack:

```
sysadmin dump_thread_stacks, RSI



T. 2006/10/23 15:37:39. (259): RS Thread Type  = 'RSI'
T. 2006/10/23 15:37:39. (259): RS Thread State =
   'Awaiting Wakeup'
T. 2006/10/23 15:37:39. (259): RS Thread Info  =
   'ost_columbia_02'
T. 2006/10/23 15:37:39. (259): Open Server Process ID:
   50, SRV_PROC address 0xed79c8
T. 2006/10/23 15:37:39. (259): Start of stack trace for
   spid 50.
T. 2006/10/23 15:37:39. (259): Native thread #70,
   FramePointer: 0xfe34f050
T. 2006/10/23 15:37:39. (259): 0x00362fc8
   sqm_read_message     (0x3345ed0, 0xfe34fdf4, 0xea60,
   0x0, 0xfe34fdf0, 0x47105f0) +0x48
T. 2006/10/23 15:37:39. (259): 0x00300908
   _rsi_sender_wrapper  (0x30c390, 0x30c230, 0x476f1f0,
   0x47105f0, 0x1f2, 0x47105f0) +0x2f28
T. 2006/10/23 15:37:39. (259): 0x002fe960
   _rsi_sender_wrapper  (0x1d794f0, 0xffffd8f1,
   0x268d14, 0xffffd800, 0x800, 0x0) +0xf80
T. 2006/10/23 15:37:39. (259): 0x0054dabc
   srv__start_function  (0xed79c8, 0x0, 0x800,
   0x862a04, 0x0, 0x0) +0x1c0
T. 2006/10/23 15:37:39. (259): 0xff265d48 _resume_ret
   (0x0, 0x0, 0x0, 0x0, 0x0, 0x0) +0x2d0
T. 2006/10/23 15:37:39. (259): End of stack trace for
   spid 50.
T. 2006/10/23 15:37:39. (259):
```

Usage

- Use sysadmin dump_thread_stack to check the internal processes of Replication Server when Replication Server is unusually slow.
- sysadmin dump_thread_stack is available for these platforms:

- Sun Solaris
- HPUX
- Linux
- IBM

Permissions — sysadmin dump_thread_stack requires “sa” permission.

See also — srv_dbg_stack() in *Open Server Server-Library/C Reference Manual*

sysadmin erssd

Description — Allows you to check ERSSD file locations and backup configurations, or perform an unscheduled backup of the ERSSD.

The command returns the status of ERSSD, including:

- ERSSD name
- Database file location
- Transaction log file location
- Transaction mirror location
- Backup start time, start date, and intervals
- Backup directory location

Syntax

```
sysadmin erssd [, backup | dbfile_dir, ’path’ | translog_dir, ’path’ |
logmirror_dir, ’path’ | defrag]
```

Parameters

backup
: Performs a single unscheduled backup of the ERSSD.

dbfile_dir, '*path*'
: Specifies a new directory for the ERSSD database file.

translog_dir, '*path*'
: Specifies a new directory for the transaction log file.

logmirror_dir, '*path*'
: Specifies a new directory for the transaction log mirror file.

defrag
: Removes fragments from the database file.

path
The pathname of the new directory.

Note Use these directory path alteration options with caution. Executing sysadmin erssd with these options automatically reboots ERSSD, and may cause system disruption.

Examples

This example shows the output of sysadmin erssd:

```
sysadmin erssd
---------------------



ERSSD Name  ERSSD Database File   ERSSD Transaction Log
----------  --------------------  ------------------
erssd.db    /dbfile/erssd.db      /log/erssd.log


ERSSD Transaction Log Mirror ERSSD Backup Start Time
---------------------------- ------------------------
/backup/erssd.mlg            2am

ERSSD Backup Start Date   ERSSD Backup Interval
-----------------------   ----------------------
March 20, 2003            12 hours

ERSSD Backup Location
---------------------
/backup
```

Usage

- Using this command with no options displays the database file path, the transaction log path, the transaction log mirror path, and the start-time, start-date, and location of scheduled transactions.
- Using this command with the backup option performs one unscheduled backup.
- Using this command with the option dbfile_dir shuts down ERSSD, moves the database to the new directory, updates the Replication Server configuration file, and restarts ERSSD, using the database from the new location.

- Using this command with the option translog_dir shuts down ERSSD, moves the transaction log file to the new directory, updates the ERSSD to use the transaction log mirror in the new direcotry, updates the Replication Server configuration file, and restarts ERSSD.
- Using this command with the option logmirror_dir shuts down ERSSD, moves the transaction log mirror file to the new directory, updates the ERSSD to use the transaction log mirror in the new directory, updates the Replication Server configuration file, and restarts ERSSD.
- Use this command with the option defrag shuts down ERSSD, rebuilds the database file, and restarts ERSSD.
- Using this command with the options defrag, dbfile_dir, translog_dir, and logmirror_dir is expensive. During this operation ERSSD is unavailable and all threads that attempt to access it fail. These threads remain blocked until ERSSD is restarted.
- Your site version must be 15.0 or above to use defrag. The defragmented file is automatically upgraded to Adaptive Server Anywhere 9.0.1 by this option, and cannot be downgraded after the command is executed.
- Use this command when you need to move files to larger, faster disks.
- Use single, not double, quotation marks in *path*.

Permissions

You must have "sa" privileges to execute this command.

sysadmin fast_route_upgrade

Description

Updates the route version to the site version of the lower of the primary or replicate Replication Server.

Upgrading a route rematerializes the data in system tables and makes information associated with new features available to a newly upgraded Replication Server.

Note Use sysadmin fast-route-upgrade *only* if the primary Replication Server has not used new features that require materialization.

Syntax

sysadmin fast_route_upgrade, *dest_replication_server*

Parameters

dest_replication_server
The destination Replication Server for the route.

Examples

Example 1 In these examples, the site version of TOKYO_RS is 1200. SYDNEY_RS has just been upgraded from 11.5 to 12.0; its site version is 1200.Issued at the source Replication Server (SYDNEY_RS) for the route terminating at the Tokyo Replication Server (TOKYO_RS), this command sets the version of the route to 12.0. New features have not yet been used at SYDNEY_RS:

```
sysadmin fast_route_upgrade, TOKYO_RS
```

Example 2 Issued at the source Replication Server (TOKYO_RS) for the route terminating at the Sydney Replication Server (SYDNEY_RS), this command is rejected since new features have been used at TOKYO_RS, and you must upgrade the route using Sybase Central’s Replication Manager plug-in:

```
sysadmin fast_route_upgrade, SYDNEY_RS
```

Usage

- Whenever Replication Servers at both ends of a route have been upgraded and site versions set to 11.5 or later, you *must* upgrade each route that connects the two servers to enable new features to flow through it. Issue this command at the source Replication Server to update the route version.
- Use sysadmin fast_route_upgrade to upgrade the route if new features have not been used at the source Replication Server.
- If you have used new features at the source Replication Server, the command is rejected and you must upgrade the route using Replication Manager (RM).

Permissions

sysadmin fast_route_upgrade requires “sa” permission.

See also

admin show_route_versions, admin show_site_version, sysadmin site_version

sysadmin hibernate_off

Description

Turns off hibernation mode for the Replication Server and returns it to an active state.

Syntax

sysadmin hibernate_off [, *string_ID*]

Parameters

string_ID

A valid identifier. If *string_ID* was specified with sysadmin hibernate_on, you must specify the same one that was used for sysadmin hibernate_on.

If you forget the *string_ID*, you can find it in the text column of the rs_recovery system table.

If you need to turn off hibernation mode for a replicate Replication Server after a successful route upgrade or route upgrade recovery, use the Replication Server name for the *string_ID*.

Examples

This command turns off the hibernation mode of the Replication Server (TOKYO_RS):

```
sysadmin hibernate_off, TOKYO_RS
```

Usage

- Hibernation mode is a Replication Server state in which:
  - all Data Definition Language (DDL) commands are rejected,
  - most service threads, such as Data Server Interface (DSI), distributor, and Replication Server Interface (RSI) sender threads, are suspended,
  - all routes and connections are suspended, and
  - RSI users are logged off and not allowed to log back into the Replication Server.
- You can execute system information (admin) and system administration (sysadmin) type commands while in hibernation mode.
- Execute this command at the Replication Server for which you want turn off hibernation mode.
- A destination Replication Server might be in hibernation mode when route upgrade fails. Do not use sysadmin hibernate_off to reactivate the Replication Server. Use Replication Manger to recover the route upgrade. For more information, please see the Replication Manger online help.
- Occasionally, a destination Replication Server is placed into hibernation mode after a successful route upgrade. Use sysadmin hibernate_off to reactivate the destination Replication Server.

Permissions

sysadmin hibernate_off requires “sa” permission.

See also sysadmin hibernate_on

sysadmin hibernate_on

Description

Turns on hibernation mode for (or suspends) the Replication Server.

Syntax

sysadmin hibernate_on [, *string_ID*]

Parameters

string_ID

A valid identifier. You must use the same *string_ID* when you execute sysadmin hibernate_off. You can use *string_ID* to ensure that no-one else accidentally turns off hibernation mode for the Replication Server while you are working on it.

If you forget the *string_ID*, you can find it in the text column of the rs_recovery system table.

Examples

This command turns on the hibernation mode of the Replication Server (TOKYO_RS):

```
sysadmin hibernate_on, TOKYO_RS
```

Usage

- Hibernation mode is a Replication Server state in which:
  - all Data Definition Language (DDL) commands are rejected,
  - most service threads, such as Data Server Interface (DSI), distributor, and Replication Server Interface (RSI) sender threads, are suspended,
  - all routes and connections are suspended, and
  - RSI users are logged off and not allowed to log back into the Replication Server.
- You can execute system information (admin) and system administration (sysadmin) type commands while in hibernation mode.
- Execute this command at the Replication Server for which you want turn on hibernation mode.
- You can turn hibernation mode on for a Replication Server to help you debug problems.

Permissions

sysadmin hibernate_on requires “sa” permission.

See also

sysadmin hibernate_off

sysadmin log_first_tran

Description
Writes the first transaction in a DSI queue into the exceptions log.

Syntax
sysadmin log_first_tran, *data_server*, *database*

Parameters
data_server
The name of the data server with the database.

database
The name of the database from whose DSI queue the first transaction is to be written.

Examples
Writes the first transaction in this DSI queue to the exceptions log:

```
sysadmin log_first_tran, SYDNEY_DS, pubs2
```

Usage

- Use sysadmin log_first_tran to write the first transaction in a DSI queue into the exceptions log.
- This command does not delete the first transaction from the queue.
- The exceptions log consists of three tables, rs_exceptshdr, rs_exceptscmd, and rs_systext. See Chapter 8, “Replication Server System Tables,” for detailed descriptions of these tables.

Permissions
sysadmin log_first_tran requires “sa” permission.

See also
admin who

sysadmin purge_all_open

Description

Purges all open transactions from an inbound queue of a Replication Server.

Syntax

sysadmin purge_all_open, *q_number*, *q_type*

Parameters

q_number, *q_type*
: Identifies the stable queue to purge. Find these values using admin who, admin who, sqm, and admin who, sqt.

Examples

Purges all open transactions from queue 103:1:

```
sysadmin purge_all_open, 103, 1
```

Usage

- Use sysadmin purge_all_open to purge all open transactions from an inbound queue of a Replication Server. Open transactions can only be purged from inbound queues.

  Note A transaction is open when the RepAgent has forwarded the transaction begin record, and possibly some commands within the transaction, but has not yet forwarded the transaction commit or abort record.

- sysadmin purge_all_open is useful if you have to truncate a data server log before it has been completely forwarded to the Replication Server, leaving open transactions in the Replication Server inbound queues. These must be removed explicitly using sysadmin purge_all_open.

  Warning! Use sysadmin purge_all_open *only* when there are open transactions in the inbound queue and you are certain that the RepAgent will not forward the commit or abort record from the log.

- Replication Server needs enough storage to purge a stable queue. If you do not have enough storage, this error message appears:

  ```
  This RS is out of Disk Space. Use another session to
  add disk space for this command to proceed.
  ```

  If this occurs, start another isql session and add stable storage to the Replication Server. sysadmin purge_all_open cannot proceed until sufficient storage is available.

- To review the contents of the transactions being dropped, execute sysadmin sqt_dump_queue before you use this command.

- If the queue has no open transactions, this command leaves the queue unchanged. If the Replication Server is restarted after transactions are purged, they may reappear as a result of recovery operations.

Permissions

sysadmin purge_all_open requires “sa” permission.

See also

admin who, alter partition, create partition, sysadmin purge_first_open, sysadmin sqt_dump_queue

sysadmin purge_first_open

Description

Purges the first open transaction from the inbound queue of a Replication Server.

Syntax

sysadmin purge_first_open, *q_number*, *q_type*

Parameters

q_number, *q_type*
Identifies the stable queue to be purged. Find these values using admin who, admin who, sqm, and admin who, sqt.

Examples

Purges the first open transaction from queue 103:1:

```
sysadmin purge_first_open, 103, 1
```

Usage

- sysadmin purge_first_open removes the first open transaction from a Replication Server's inbound queue. RepAgent threads transfer transactions from the database log one record at a time. A transaction is open when the RepAgent has forwarded the transaction begin record, and possibly some commands within the transaction, but has not yet forwarded the transaction commit or abort record.
- sysadmin purge_first_open can be only used with inbound queues.
- Replication Server needs enough space to purge the first open transaction from a stable queue. If there is not enough disk space, this error message appears:

  ```
  This RS is out of Disk Space. Use another session to
  add disk space for this command to proceed.
  ```

  If this occurs, start another isql session and add stable storage (disk space) to the Replication Server. sysadmin purge_first_open cannot proceed until sufficient storage is available.
- To review the contents of the transaction being dropped, execute sysadmin sqt_dump_queue before you use this command.
- To display information about the first transaction in the inbound queue, use admin who, sqt. If the state of the first transaction is “open” (ST:O), it can be dropped from the queue.
- The sysadmin purge_first_open command is useful when there is an uncommitted transaction in the Adaptive Server log. The open transaction is delivered by the RepAgent to Replication Server. Because there is an open transaction, Replication Server cannot truncate the inbound queue. If the transaction remains open for a long time, the inbound queue fills and Replication Server may run out of queue space.

- If the first transaction of the queue is not open, this command leaves the queue unchanged. If the Replication Server is restarted after a transaction is dropped, the transaction may reappear as a result of recovery operations.

Warning! Use sysadmin purge_first_open only when you have determined (by using admin who, sqt and admin who, sqm) that the inbound queue is stuck on an uncommitted transaction.

Permissions

sysadmin purge_first_open requires "sa" permission.

See also

admin who, alter partition, create partition, sysadmin dump_queue, sysadmin purge_all_open

sysadmin purge_route_at_replicate

Description

Removes all references to a primary Replication Server from a replicate Replication Server.

Syntax

sysadmin purge_route_at_replicate, *replication_server*

Parameters

replication_server
The name of the primary Replication Server to be purged from the replicate's RSSD.

Examples

Purges the primary Replication Server, TOKYO_RS, from the replicate's RSSD:

```
sysadmin purge_route_at_replicate, TOKYO_RS
```

Usage

- Use sysadmin purge_route_at_replicate to remove all subscriptions and route information originating from a specified primary Replication Server after the route is dropped from it. This is useful after drop route with nowait is executed at the primary Replication Server.
- If there is a route from the current Replication Server to the specified primary Replication Server, you must drop the route before executing this command.
- If a subscription was materializing when drop route with nowait was executed at the primary Replication Server, a materialization queue may be left at the replicate Replication Server. Use sysadmin drop_queue to remove this queue.

Warning! Use sysadmin purge_route_at_replicate only if the drop route with nowait command was executed at the primary Replication Server or if the primary Replication Server is lost and will not be recovered.

Permissions

sysadmin purge_route_at_replicate requires "sa" permission.

See also

drop route, rs_helproute

sysadmin restore_dsi_saved_segments

Description
Restores backlogged transactions.

Syntax
sysadmin restore_dsi_saved_segments, *data_server*, *database*

Parameters
data_server
The name of the data server.

database
The name of the database.

Examples
Restores backlogged transactions for the pubs2 database in the TOKYO_DS data server:

```
sysadmin restore_dsi_saved_segments, TOKYO_DS, pubs2
```

Usage

- The DSI must be explicitly suspended before you can use this command to restore saved segments.
- Any backlogged transactions saved because a save interval was specified for the connection (using alter connection) are candidates for restoring into the database. The Replication Server uses rs_get_lastcommit to decide which transactions to filter.

Permissions
sysadmin restore_dsi_saved_segments requires "sa" permission.

See also
configure connection

sysadmin set_dsi_generation

Description	Changes a database generation number in the Replication Server to prevent the application of transactions in the DSI stable queue after a replicate database is restored.
Syntax	sysadmin set_dsi_generation, *gen_number*, *primary_data_server*, *primary_database*, *replicate_data_server*, *replicate_database*

Parameters

gen_number
The new generation number of the database. The number is an integer between 0 and 65,535.

primary_data_server
The name of the data server at the primary site.

primary_database
The name of the primary database.

replicate_data_server
The name of the replicate data server.

replicate_database
The name of the replicate database.

Examples

Sets new DSI generation number to 105. The previous number was 104 or less:

```
sysadmin set_dsi_generation 105 NY_DS, ny_db, SF_DS,
         sf_db
```

Usage

Use sysadmin set_dsi_generation during the recovery of a database dump. Changing the generation number except during recovery may cause incorrect data at replicate databases.

See the *Replication Server Administration Guide Volume 2* for a complete description of the recovery procedure.

Permissions

sysadmin set_dsi_generation requires “sa” permission.

See also

admin get_generation, configure connection, dbcc dbrepair, dbcc settrunc, rebuild queues

sysadmin site_version

Description

Sets the site version number for the Replication Server. This lets you use the software features in the corresponding release, and prevents you from downgrading to an earlier release. If the Replication Server uses ERSSD, this command also shuts down the ERSSD, upgrades its database file and restarts ERSSD.

Note If your Replication Server uses ERSSD, this command may cause some threads to shutdown since ERSSD is being restarted. Replication should continue after you restart all threads that are shutdown.

Syntax

sysadmin site_version [, version]

Parameters

version
The site version number for Replication Server.

Version number	Site version
Pre-11.5	N/A
11.5	1150
12.0	1200
12.5	1250
12.6	1260
15.0, 15.0.1	1500

No site version numbers exist for releases earlier than 11.5. Maintenance releases may support higher site version numbers.

Examples

Example 1 Displays the current site version number for the Replication Server:

```
sysadmin site_version
```

Example 2 Changes the site version number to correspond to release 15.0:

```
sysadmin site_version, 1500
```

Usage

- To set the site version number for the current Replication Server, execute sysadmin site_version with a *version* parameter.

  The site version number you enter must be no higher than the software version number or the release level of Replication Server.

- To display the site version number for Replication Server, execute sysadmin site_version without a *version* parameter.

System version and site version

- Starting with Replication Server version 11.5, you can use new software features up to the version set in Replication Server's site version.

  A minimum system version number of 1102 is required. See sysadmin system_version for more information.

- The initial site version number for a Replication Server depends on whether you have installed a new Replication Server or upgraded from an earlier release:

  - For a newly installed Replication Server of release 15.0, the site version number is 1500.
  - If you have upgraded to release 11.5 or higher from release 10.1.x or 11.0.x, the site version number matches the system version number. You can use sysadmin site_version to set the site version number to 1150 or higher only if the system version number is set to 1102.

- For more information about features that were introduced in a particular Replication Server software release, see *What's New in Replication Server* for that release.

Warning! When you set the site version number, you cannot downgrade to an earlier release.

- For more information about installing or upgrading Replication Servers, refer to the Replication Server installation and configuration guides for your platform.

Mixed-version replication systems

- In a mixed-version replication system, an 1102 system version number and site version numbers for individual Replication Servers allow release 11.0.2 or 11.0.3 Replication Servers to work with release 11.5 or higher Replication Servers.

  In such a system, new features may not be available to earlier versions of Replication Server. For example, a Replication Server of release 11.5 can create multiple replication definitions for a table. An 11.0.2 or 11.0.3 Replication Server can subscribe to only one such replication definition. See the *Replication Server Administration Guide Volume 1* for more information.

Upgrading routes

- After you have upgraded one or both Replication Servers on either end of a route to release 11.5 or later, and you have set the site versions to a higher level, you need to upgrade the route. Upgrading a route rematerializes the data in system tables and makes information associated with new features available to a newly upgraded Replication Server.

  There are two possible scenarios for route upgrade:

  - If you have Replication Manger, use the Replication Manger to upgrade routes. For instruction on upgrading routes, please see the Replication Manger online help
  - If new features have not been used at the source Replication Server, use sysadmin fast_route_upgrade to upgrade routes.

  For example, if you upgrade a Replication Server of release 12.6 to release 15.0 and set its site version accordingly, you will need to upgrade a route from another Replication Server of release 15.0. When you upgrade the route, the newly upgraded Replication Server receives information from the 15.0 Replication Server such as additional replication definitions for the table.

  See the *Replication Server Configuration Guide* for more information about upgrading routes.

System tables for version information

Version information is stored in the rs_version system table. The rs_routes system table also contains version information. Route version information is stored in the *rs_routeversions* system table.

Permissions

sysadmin site_version requires “sa” permission.

See also

admin version, sysadmin fast_route_upgrade, sysadmin system_version

sysadmin sqm_purge_queue

Description

Purges all messages from a stable queue.

Warning! Purging messages from a stable queue can result in data loss and should be used only with the advice of Sybase Technical Support. Replication Server cannot send purged messages to the destination database or Replication Server, and this causes inconsistencies in the replication system. If a queue contains subscription marker messages or route messages, using this command can have severe consequences.

Syntax

sysadmin sqm_purge_queue, *q_number*, *q_type*

Parameters

q_number, *q_type*
Identifies the stable queue to be purged. Find these using admin who, admin who, sqm, or admin who, sqt.

Examples

Purges all messages from inbound queue number 103:

```
sysadmin sqm_purge_queue, 103, 1
```

Usage

- sysadmin sqm_purge_queue removes messages destined to another Replication Server from a stable queue. Use this command when your queues are filled with messages.
- sysadmin sqm_purge_queue can only be executed when the Replication Server has been started in standalone mode.

Permissions

Requires “sa” permission.

See also

admin who, repserver

sysadmin sqm_unzap_command

Description
Undeletes a message in a stable queue.

Syntax
sysadmin sqm_unzap_command, *q_number*, *q_type*,
seg, *blk*, *row*

Parameters
q_number, *q_type*
Identifies the stable queue with the message to be undeleted. Find these values using admin who, admin who, sqm, and admin who, sqt.

seg
Identifies the segment in the stable queue that contains the message to be undeleted.

blk
Identifies the 16K block in the segment. Block numbering starts at 1 and ends at 64.

row
The row number in the block of the command to be undeleted.

Usage

- The Replication Server must be in standalone mode to use sysadmin sqm_unzap_command.
- sysadmin sqm_unzap_command removes the delete mark from a message in a stable queue. Use this command to undelete a message that you marked deleted using sysadmin sqm_zap_command.
- Use sysadmin dump_queue to locate the message you want to undelete.

Permissions
sysadmin sqm_unzap_command requires “sa” permission.

See also
admin who, sysadmin drop_queue, sysadmin sqm_zap_command

sysadmin sqm_zap_command

Description
: Deletes a single message in a stable queue.

Syntax
: sysadmin sqm_zap_command, *q_number*, *q_type*,
 seg, *blk*, *row*

Parameters

q_number, *q_type*
: Identifies the stable queue with the message to be deleted. Find these values using admin who, admin who, sqm, and admin who, sqt.

seg
: Identifies the segment in the stable queue.

blk
: Identifies the 16K block in the segment. Block numbering starts at 1 and ends at 64.

row
: The row number in the block of the command to be deleted.

Examples

```
sysadmin sqm_zap_command
```

```
sysadmin sqm_zap_command, 103, 1, 15, 65, 2
```

Usage

- The Replication Server must be in standalone mode to use sysadmin sqm_zap_command.
- Use sysadmin dump_queue to locate the message you want to delete.
- sysadmin sqm_zap_command marks a message in a stable queue as deleted. When Replication Server processes the queue, it ignores the marked message.
- You can undelete a message using sysadmin sqm_unzap_command. This command removes the delete mark from the message.
- If you delete a message and then restart Replication Server in normal mode, the part of the queue holding the message may have been processed. If it has, you cannot to undelete the message with sqm_unzap_command.

Permissions
: sysadmin sqm_zap_command requires “sa” permission.

See also
: admin who, sysadmin dump_queue, sysadmin sqm_unzap_command

sysadmin sqt_dump_queue

Description

Dumps the transaction cache for an inbound queue or a DSI queue.

Syntax

sysadmin sqt_dump_queue, *q_number*, *q_type*, *reader*
[, open]

Parameters

q_number, *q_type*
Identifies the stable queue to be dumped. Find these values using admin who, admin who, sqm, and admin who, sqt.

reader
Identifies the reader you want to dump the stable queue for. This parameter applies to features that require multiple readers, such as warm standby applications. You can get the reader number from admin sqm_readers or from admin who, sqt. If you are not using multiple readers, enter “0” for the reader.

open
Allows you to dump only open transactions. If you use this option, insert a comma between *q_type* and the open flag.

Examples

Dumps all undeleted transactions in queue 103:1 from the transaction cache:

```
sysadmin sqt_dump_queue, 103, 1, 0
```

Usage

- Before using sysadmin sqt_dump_queue, execute admin who, sqt to make sure the transaction cache for the database exists.
- This command dumps all the statements of transactions in the transaction cache.
- Transaction statements are dumped into the Replication Server log or an alternate log file specified with sysadmin dump_file.
- The output from this command indicates the state of transactions in the transaction cache as open, closed, or read. Open transactions do not have a commit yet. Closed transactions have a commit but have not been completely read out yet. Read transactions have been completely read out but have not been deleted yet.
- You can modify the cache size by setting the configuration parameter sqt_max_cache_size.

Permissions

sysadmin sqt_dump_queue requires “sa” permission.

See also

admin who, sysadmin dump_file

sysadmin system_version

Description

Displays or sets the system-wide version number for the replication system, allowing you to use the software features in the corresponding release level.

Starting with release 11.5, the site version for individual Replication Servers also enables new features. The system version number need not correspond to the current software version.

Syntax

sysadmin system_version [, *version*]

Parameters

version

The system version number to use for the replication system.

Valid system version numbers include 1200, 1210, 1250, 1260, and 1500, which correspond to Replication Server versions. Maintenance releases may support higher system version numbers.

Refer to the Replication Server release bulletin for your platform for information about supported system version numbers.

Examples

Example 1 Executed at the ID Server, displays the current system version number:

```
sysadmin system_version
```

Example 2 Executed at the ID Server, changes the system version number to correspond to release 15.0. You can use this number if:

- All Replication Servers are at release 15.0
- You will not need to downgrade any Replication Servers to an earlier release
- You will not need to install any Replication Servers of an earlier release

```
sysadmin system_version, 1500
```

Usage

- To set the system version number, execute sysadmin system_version at the ID Server, and include a *version* parameter.
  - The system version number you enter must be no higher than the lowest software version number—the release level of a Replication Server—of any Replication Server in the replication system.
  - You cannot set the system version number at any other Replication Server than the ID Server.
- To display the current system version number, execute sysadmin system_version at the ID Server, without a *version* parameter.

If you execute this command at another Replication Server, the Replication Server tries to contact the ID Server to determine the current system version number. In rare cases, a Replication Server may be unable to contact the ID Server. For this reason, only the value at the ID Server is guaranteed to be correct.

System version number and 11.0.*x* and earlier releases

- For Replication Server releases through 11.0.*x*, the system version number lets you use Replication Server features through the corresponding release level. You set the system version to a number corresponding to the lowest release level of any Replication Server in the replication system.

  The system version number allows newer releases of Replication Server to work in a limited, lowest common denominator fashion with older releases of Replication Server.

System version and site version

- Starting with Replication Server release 11.5, you can use certain new software features when the Replication Server's site version number has been set to the current software release—for example, 1150 for release 11.5. See sysadmin site_version for more information.

  A minimum system version number of 1102 is also required.

- When you install a Replication Server of release 11.5 or higher as the ID Server for a new replication system, the system version number is set to 1102. This number allows you to install additional Replication Servers of release 11.0.2 or later into the system.

- When you install a Replication Server of release 11.5 or higher into an existing replication system, or upgrade an existing Replication Server of release 10.1.x or 11.0.x, the system version number remains unchanged. New 11.5 or higher features cannot be used until the site version is set to 1150 or higher and the system version is set to 11.0.2 or higher. See sysadmin site_version for more information.

  Once all the Replication Servers have been upgraded to a higher minimum level, use sysadmin system_version to set the system version number to a higher level.

Warning! Once you have set the system version number, you cannot downgrade any Replication Servers to an earlier release level or install Replication Servers of an earlier release level.

- For more information about installing or upgrading Replication Servers, refer to the Replication Server installation and configuration guides for your platform.

Mixed-version replication systems

If all of your Replication Servers are at release 11.0.2 or later, the highest required setting for the system version number is 1102. After setting the system version number to 1102, you may never need to set it again.

A 1102 system version number and site version numbers for individual Replication Servers allows a mixed-version replication system, in which Replication Servers of release 11.0.2 or 11.0.3 or later can work together with Replication Servers of release 11.5 or later. Each Replication Server can use its full set of available features. See sysadmin site_version for more information.

For more information about features that were introduced in a particular Replication Server software release, see *What's New in Replication Server* for that release.

System version and the ID Server

In Replication Servers other than the ID Server, when a command is executed that requires a certain minimum system version, the Replication Server contacts the ID Server to determine the current system version number before allowing use of the command.

For example, create function replication definition was introduced at release 11.0, and requires a minimum system version number of 1100. If the system version number is at 1011, corresponding to release 10.1.1, you cannot use this command.

System tables for version information

Version information is stored in the rs_version system table. The rs_routes system table also contains version information.

Permissions

sysadmin system_version requires "sa" permission.

See also

admin version, sysadmin site_version

validate publication

Description

Sets the status of a publication to VALID, allowing new subscriptions to be created for the publication.

Syntax

validate publication *pub_name*
with primary at *data_server.database*

Parameters

pub_name
The name of the publication to be validated.

with primary at data_server.database
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

Examples

Validates the publication pubs2_pub:

```
validate publication pubs2_pub
    with primary at TOKYO_DS.pubs2
```

Usage

- When all of the articles have been created for a publication, you must validate the publication using validate publication before a replicate site can subscribe to it. Validating a publication verifies that the publication contains at least one article and marks the publication ready for subscription.
- Execute validate publication at the Replication Server where you created the publication using create publication.
- To check the status of a publication, use check publication. This command displays the number of articles the publication contains and indicates if the publication is valid.

See the *Replication Server Administration Guide Volume 1 and Volume 2* for more information about subscription materialization.

Permissions

validate publication requires "create object" permission.

See also

check publication, check subscription, create publication, create subscription, define subscription, drop publication

validate subscription

Description

For a subscription to a replication definition or a publication, sets the subscription status to VALID. This command is part of the bulk materialization process, or part of the process of refreshing a publication subscription.

Syntax

validate subscription *sub_name*
for {*table_rep_def* | *function_rep_def* |
  publication *pub_name* | database replication definition *db_repdef*
  with primary at *data_server.database* }
with replicate at *data_server.database*

Parameters

sub_name
The name of the subscription to be validated.

for *table_rep_def*
Specifies the name of the table replication definition the subscription is for.

for *function_rep_def*
Specifies the name of the function replication definition the subscription is for.

for publication *pub_name*
Specifies the name of the publication the subscription is for.

for database replication definition *db_repdef*
Specifies the name of the database replication definition the subscription is for.

with primary at *data_server.database*
Specifies the location of the primary data. If the primary database is part of a warm standby application, *data_server.database* is the name of the logical data server and database. Include this clause only for a subscription for a publication.

with replicate at *data_server.database*
Specifies the location of the replicate data. If the replicate database is part of a warm standby application, *data_server.database* is the name of the logical data server and database.

Examples

Example 1 Validates the subscription titles_sub for the table replication definition titles_rep, where the replicate database is SYDNEY_DS.pubs2:

```
validate subscription titles_sub
 for titles_rep
 with replicate at SYDNEY_DS.pubs2
```

Example 2 Validates the subscription myproc_sub for the function replication definition myproc_rep, where the replicate database is SYDNEY_DS.pubs2:

```
validate subscription myproc_sub
 for myproc_rep
 with replicate at SYDNEY_DS.pubs2
```

Example 3 Validates the subscription pubs2_sub for the publication pubs2_pub, where the primary database is TOKYO_DS.pubs2 and the replicate database is SYDNEY_DS.pubs2:

```
validate subscription pubs2_sub
 for publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replicate at SYDNEY_DS.pubs2
```

Usage

- Use validate subscription to validate a subscription at the primary and replicate Replication Servers. The subscription can be to a table replication definition, function definition replication, or publication.
- This command completes the bulk materialization process. The first step is creating the subscription using define subscription. The second step is activating the subscription using activate subscription.
- If you have added any new articles to a publication with an existing subscription, you must refresh the publication subscription in order to create new subscriptions for these articles.

  Use define subscription and activate subscription to create and activate the new article subscriptions in the publication subscription. Then manually load the subscription data for the new article subscriptions, and use validate subscription to validate the publication subscription.
- Execute validate subscription at the Replication Server where you created the subscription using define subscription.
- When you validate a publication subscription, all of its article subscriptions are validated at the same time.
- validate subscription changes the status of a subscription from ACTIVE to VALID. Subsequent updates at the primary data server are distributed through the primary Replication Server and applied at the replicate Replication Server.
- This command modifies RSSD tables at multiple sites. Use check subscription at both the primary and replicate Replication Servers to see the effects on each.

See the *Replication Server Administration Guide Volume 1 and Volume 2* for more information about subscription materialization.

Permissions

validate subscription requires “create object” permission at the site where the data is replicated and “primary subscribe” or “create object” permission at the site where the primary data is stored.

See also

activate subscription, check subscription, create article, create publication, create subscription, define subscription, drop subscription

wait for create standby

Description	A blocking command that allows a client session in the Replication Server to wait for the standby database creation process to complete.

Syntax

```
wait for create standby
 for logical_ds.logical_db
```

Parameters

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

Usage

- After the standby database has been created, wait for create standby displays status information.
- wait for create standby may be most helpful when used in scripts.

Permissions

wait for create standby requires “sa” permission.

See also

abort switch, switch active, wait for switch

wait for delay

Description	Specifies a time interval at which this command is blocked.
Syntax	wait for delay ’time_string’
Parameters	*time_string* The period of time passed before executing. Uses the format hh:mm[:ss[.xxx]] [am|pm].
Examples	This command instructs Replication Server to block a command for 1 hour and 30 minutes:

```
wait for delay '01:30'
```

Usage

- Use wait for delay to instruct Replication Server to wait until the specified period of time has passed. A typical usage is in implementing subscriptions. Usually, wait for delay is issued in between two subscriptions.
- The time specified can include hours, minutes, and seconds, up to a maximum of 24 hours.

Permissions	Any user can execute this command.
See also	wait for time

wait for switch

Description

A blocking command that allows a client session in the Replication Server to wait for the switch to the new active database to complete.

Syntax

```
wait for switch
 for logical_ds.logical_db
```

Parameters

logical_ds
The data server name for the logical connection.

logical_db
The database name for the logical connection.

Usage

- After the switch active operation is complete, wait for switch displays status information.
- wait for switch may be most helpful when used in scripts.

Permissions

wait for switch requires “sa” permission.

See also

abort switch, switch active, wait for create standby

wait for time

Description	Specifies a time of day at which to unblock this command.
Syntax	wait for time 'time_string'
Parameters	*time_string* The specific time to execute. Uses the format hh:mm[:ss[.xxx]] [am|pm].

Examples

This command instructs Replication Server to wait until 5:30 p.m.:

```
wait for time '05:30 pm'
```

Usage

- Use wait for time to instruct Replication Server to wait until the specified time.
- The time specified can include hours, minutes, and seconds, up to a maximum of 24 hours.

  If the current time is 6:00 pm, wait for time '5:00 pm' indicates 5:00 p.m. tomorrow.

Permissions

Any user can execute this command.

See also

wait for delay

CHAPTER 4 Replication Server System Functions

This chapter contains reference pages for the Replication Server system functions.

See the *Replication Server Administration Guide Volume 2*, for information about customizing function strings for system functions.

The system functions described in this chapter may have **function-string-class scope** or **replication-definition scope**.

A function that has function-string class scope is defined once, for its class. It is then applied the same way in every database to which the class is assigned.

A function that has replication definition scope is defined once for each replication definition. It is then applied the same way for every operation (update, insert, and so on) that is replicated using the replication definition.

rs_batch_end

Description

rs_batch_end allows users to batch commands into non-Adaptive Server database servers. This function string stores the SQL statements needed to mark the end of a batch of commands.

Examples

Alters rs_batch_end function string so that the SQL output of the function-string class sqlserver_derived_class is END.

```
alter function string publishers.rs_batch_end
for sqlserver_derived_class
output language
'END'
```

Usage

- The rs_batch_end function has function-string class scope.
- This function string is used with rs_batch_start.
- rs_batch_end is sent to the replicate data server as the last command in the batch of commands. It is sent only if use_batch_markers is set to on.

- rs_batch_end precedes rs_commit in the order of data server processing.
- rs_batch_start, a batch of commands, and rs_batch_end may be repeated for a given transaction if more than one batch is required due to commands being flushed by limits such as dsi_cmd_batch_size.

See also rs_batch_start

rs_batch_start

Description

rs_batch_start allows users to batch commands into non-Adaptive Server database servers. This function string stores the SQL statements needed to mark the beginning of a batch of commands.

Examples

Alters rs_batch_start function string so that the SQL output of the function-string class sqlserver_derived_class is `BEGIN`.

```
alter function string publishers.rs_batch_start
for sqlserver_derived_class
output language
'BEGIN'
```

Usage

- The rs_batch_start function has function-string-class scope.
- Use of rs_batch_start is not necessary for Adaptive Server or any other data server that supports command batching by the function strings rs_begin and rs_commit.
- rs_batch_start and the batch of commands following it is sent to the replicate data server only if use_batch_markers is set to on. rs_batch_start is sent after rs_begin.
- Replication Server does not use the command separator following rs_batch_start. If the replicate database server requires a command separator following the marker for the beginning of a batch it is included as part of the string for rs_batch_start. This separator must be included as part of the function string whether it is the same or different from the dsi_cmd_separator parameter.
- The rs_batch_start, a batch of commands, and rs_batch_end may be repeated if more than one batch is required due to commands being flushed by limits such as dsi_cmd_batch_size.

See also

rs_batch_end

rs_begin

Description

Begins a transaction in a data server.

Examples

Example 1 Creates an rs_begin function string for the oth_sql_class function-string class. The *rs_origin_xact_name* system variable has a null value if the transaction has no name. Placing "t_" in front of the system variable prevents data server syntax errors and allows the function string to support named and unnamed transactions.

```
alter function string rs_begin
 for oth_sql_class
 output language
 'begin transaction
   t_?rs_origin_xact_name!sys_raw?'
```

Example 2 Creates an rs_begin function string for a function-string class for a data server that does not support the begin transaction operation.

```
create function string rs_begin
 for oth_sql_class
 output language ''
```

Usage

- The rs_begin function has function-string-class scope.
- Replication Server creates an initial rs_begin function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_begin function string.
- Create or customize an rs_begin function string at the Replication Server that is the primary site for the class.
- Some data servers do not support an explicit begin transaction operation. Instead, they begin transactions implicitly whenever the previous transaction is committed or rolled back. For these data servers, the rs_begin function string can be an empty string (").
- The function string for this function usually uses the *rs_origin_xact_name* system variable. Its value is received from the RepAgent. The transaction name is assigned in Transact-SQL with begin transaction.

See also

alter function string, create function string, rs_commit, rs_rollback

rs_check_repl

Description	Checks to see if a table is marked for replication.

Examples

Creates an rs_check_repl function string that executes the rs_check_repl_stat stored procedure.

```
create function string rs_check_repl
 for sqlserver_derived_class
 output language
 'execute rs_check_repl_stat
  @rs_repl_name = ?rs_repl_name!param?'
```

Usage

- The rs_check_repl function has function-string-class scope.
- Replication Server creates an initial rs_check_repl function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_check_repl function string.
- Create or customize an rs_check_repl function string at the Replication Server that is the primary site for the class.

See also

create function string, create replication definition

rs_commit

Description

Commits a transaction in a data server.

Examples

This example illustrates the default rs_commit function string for the rs_sqlserver_function_class and rs_default_function_class classes. The function string executes a stored procedure named rs_update_lastcommit and then executes the Transact-SQL commit transaction command.

```
create function string rs_commit
 for sqlserver_derived_class
 output language
 'execute rs_update_lastcommit
   @origin = ?rs_origin!sys?,
   @origin_qid = ?rs_origin_qid!sys?,
   @secondary_qid = ?rs_secondary_qid!sys?,
   @origin_time = ?rs_origin_commit_time!sys?;
 commit transaction'
```

Here is the text of the rs_update_lastcommit procedure for rs_sqlserver_function_class:

```
/* Create a procedure to update the
 ** rs_lastcommit table. */
 create procedure rs_update_lastcommit
     @origin  int,
     @origin_qid  binary(36),
     @secondary_qid  binary(36),
     @origin_time  datetime
 as
 begin
     update rs_lastcommit
         set origin_qid = @origin_qid,
         secondary_qid = @secondary_qid,
         origin_time = @origin_time,
         commit_time = getdate()
     where origin = @origin
     if (@@rowcount = 0)
     begin
         insert rs_lastcommit (origin,
             origin_qid, secondary_qid,
             origin_time, commit_time,
             pad1, pad2, pad3, pad4,
             pad5, pad6, pad7, pad8)
             values (@origin, @origin_qid,
             @secondary_qid,@origin_time,
             getdate(), 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00)
```

```
          end
      end
```

Usage

- The rs_commit function has function-string-class scope.
- Replication Server creates an initial rs_commit function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_commit function string.
- Create or customize an rs_commit function string at the Replication Server that is the primary site for the class.
- Update the rs_lastcommit system table in the rs_commit function string. Updating this table within the transaction maintains data integrity.

Warning! If the rs_lastcommit system table is not updated properly for each transaction committed, after a restart Replication Server may apply transactions more than once or skip transactions.

See also

alter function string, create function string, rs_begin, rs_get_lastcommit, rs_rollback

rs_datarow_for_writetext

Description

Provides an image of the data row associated with a text, unitext, or image column updated with the Transact-SQL writetext command, with the Client-Library function ct_send_data, or with the DB-Library™ functions dbwritetext and dbmoretext.

Examples

Executes a stored procedure named capture_datarow, setting the value of *@au_id* to the value of the au_id column and the value of *@copy* to the status value for the copy column.

```
create function string
 blurbs_rep.rs_datarow_for_writetext
 for sqlserver_derived_class
 output rpc
 'execute capture_datarow
   @au_id = ?au_id!new?,
   @copy = ?copy!text_status?'
```

Usage

- Replication Server executes rs_datarow_for_writetext before updated text, unitext, or image data is sent to the replicate data server. rs_datarow_for_writetext provides the values of primary key columns and searchable columns from the row so that subscriptions can be processed and data can be transferred to the replicate database.
- rs_datarow_for_writetext accesses the values of all columns in the row except for text, unitext, and image columns. To retrieve information about text, unitext, or image columns, include the *text_status* modifier in the function string. The values returned by *text_status* are described in Table 4-1.
- The rs_datarow_for_writetext function has replication definition scope.
- Replication Server generates an rs_datarow_for_writetext function string for rs_sqlserver_function_class and rs_default_function_class when you create a replication definition.
- If you use a user-created base function-string class, you must create a rs_datarow_for_writetext function string for each replication definition that includes text, unitext, and image columns.
- Create or customize a rs_datarow_for_writetext function string at the Replication Server where you created the replication definition.
- The default generated function string for rs_sqlserver_function_class and rs_default_function_class does not execute commands in the replicate database, since the row image contains no modified data.

- You can create a new rs_datarow_for_writetext function string to collect the values of the primary key to pass to a gateway. The *old* and *new* modifiers both provide access to a column's value.
- The *text_status* modifier retrieves the status of the text, unitext, or image column. Table 4-1 lists the possible values for the *text_status* modifier.

Table 4-1: text_status values for text, unitext, and image data

Value	Description
0x0001	The column has a null text pointer. There are no modifications to text, unitext, or image columns.
0x0002	Modifications were made at the primary database, which caused a text pointer allocation. Replication Server executes the rs_textptr_init function to allocate a text pointer.
0x0004	The current data value follows. Replication Server executes the rs_writetext function to modify the text, unitext, or image data at the replicate database.
0x0008	The text, unitext, or image column is not replicated. No commands are required in the replicate database because the data did not change value and the text, unitext, or image column has a replicate_if_changed status.
0x0010	The text, unitext, or image column contains a null value after an operation at the primary database. For example, after a text pointer has been allocated, there may be data values in a text or image column and an application at the primary database sets them to null. Replication Server truncates the text, unitext, or image column in the replicate database by setting the values to null if the *text_status* is not 0x0008.

See also rs_get_textptr, rs_textptr_init, rs_writetext

rs_delete

Description

Deletes a row in a replicated table.

Examples

Changes the rs_delete function string for the titles_rep replication definition so that it executes a stored procedure named del_title.

```
alter function string titles_rep.rs_delete
 for sqlserver_derived_class
 output rpc
 'execute del_title
   @title=?title!old?'
```

Usage

- Replication Server executes rs_delete to delete a single row in a table. The row is identified by the primary key columns defined in a replication definition for the table.
- rs_delete has replication definition scope.
- Replication Server generates an rs_delete function string for the system-provided function-string classes when you create a replication definition.
- If you use a user-created base function-string class, you must create an rs_delete function string for each replication definition.
- Create or customize an rs_delete function string where you created the replication definition.
- For the system-provided classes rs_sqlserver_function_class and rs_default_function_class, the rs_delete generated function string uses the Transact-SQL delete command syntax. The row to be deleted is identified with a where clause that specifies the pre-delete values, or before image, of the primary key columns.

See also

create function string, create replication definition, rs_insert, rs_update

rs_dumpdb

Description

Initiates a coordinated database dump.

Examples

Example 1 Creates an rs_dumpdb function string that dumps the database to a specified dump device and executes a procedure to update the rs_lastcommit system table. This function string works best when there is only one replicate database or when all databases using the function-string class have the same dump device names.

```
create function string rs_dumpdb
 for sqlserver_derived_class
 output language
 'dump database ?rs_destination_db!sys_raw?
     to pubs2_dmpdb;
 execute rs_update_lastcommit
     ?rs_origin!sys?,
     ?rs_origin_qid!sys?,
     ?rs_secondary_qid!sys?,
     ?rs_origin_commit_time!sys?'
```

Example 2 This example is better suited to multiple sites and production environments than is the first example. dumpdb_proc manages the backup devices at the replicate sites. The procedure should select a backup device to use, then mark it "used" so that a subsequent dump does not overwrite the previous backup.

```
alter function string rs_dumpdb
 for sqlserver_derived_class
 output rpc
 'execute dumpdb_proc
     ?rs_dump_dbname!sys?,
     ?rs_dump_label!sys?,
     ?rs_dump_timestamp!sys?,
     ?rs_destination_db!sys?,
     ?rs_origin!sys?,
     ?rs_origin_qid!sys?,
     ?rs_secondary_qid!sys?,
     ?rs_origin_commit_time!sys?'
```

The procedure uses *rs_origin*, *rs_origin_qid*, and *rs_secondary_qid* to execute rs_update_lastcommit. If the server fails after the dump is complete but before the rs_lastcommit system table is updated, the backup is restarted when Replication Server resumes.

Note There is no guarantee that the dump and the rs_update_lastcommit procedure will execute atomically, because Adaptive Server does not allow the dump command to be included in a transaction with other commands. If the rs_lastcommit system table is not updated successfully, an additional dump may be performed.

In the following sample text of the dumpdb_proc stored procedure, the dump devices are hard-coded. In a production environment, it is better to manage them in a table.

```
create proc dumpdb_proc
         @dump_dbname varchar(30),
         @dump_label varchar(30),
         @dump_timestamp varbinary(16),
         @destination_dbname varchar(30),
         @origin int,
         @origin_qid binary(36),
         @secondary_qid binary(36),
         @origin_time datetime
 as
 print 'Received a dump database command from
Replication Server:'
 declare @message varchar(255)
 select @message = 'dump database ' + @dump_dbname
         + '. Label= '' + @dump_label
         + ''. Dest.db = '' + @destination_dbname
         + ''''
 print @message
 if @destination_dbname = 'pubs2'
 begin
         print 'issuing ''dump database pubs2.'''
         dump database pubs2 to pubs2_dmplog
         update dmp_count set d_count = d_count + 1
         exec pubs2.dbo.rs_update_lastcommit
             @origin, @origin_qid, @secondary_qid,
             @origin_time
 end
 else if @destination_dbname = 'pubs3'
 begin
         print 'issuing ''dump database pubs3.'''
```

```
            dump database pubs3 to pubs3_dmplog
            update dmp_count set d_count = d_count + 1
            exec pubs3.dbo.rs_update_lastcommit
                @origin, @origin_qid, @secondary_qid,
                @origin_time
    end
```

Usage

- Replication Server coordinates database dumps by placing rs_dumpdb function calls in the same place in the stream of transactions distributed to each replicate Replication Server.
- rs_dumpdb has function-string class scope.

Note Replication Server does not initialize or generate rs_dumpdb function strings for the system-provided function-string classes. You must create a function string before using a coordinated dump with Adaptive Server.

- Create an rs_dumpdb function string at the Replication Server that is the primary site for the class.
- To account for different dump devices at multiple replicate sites, create a stored procedure in each replicate database that performs a database dump. Then write the rs_dumpdb function string to execute the stored procedure.
- The rs_lastcommit system table should be updated when the rs_dumpdb function string executes so that a restarted Replication Server does not perform duplicate dumps. See rs_commit for information about rs_lastcommit.
- Table 4-2 lists the system variables that can be used in rs_dumpdb function strings.

Table 4-2: System variables for rs_dumpdb function strings

Variable name	Datatype	Description
rs_dump_dbname	varchar(30)	The name of the database where the dump originated.
rs_dump_label	varchar(30)	Label information for the dump. For Adaptive Server, this variable holds a datetime value that is the time the dump originated.
rs_dump_timestamp	varbinary(16)	A timestamp taken when the dump started.

See also

create function string class, rs_commit, rs_dumptran, rs_get_lastcommit

rs_dumptran

Description

Initiates a coordinated transaction dump.

Examples

Example 1 Creates an rs_dumptran function string to execute a stored procedure named dumptran_proc. The stored procedure manages the dump devices and then executes the rs_update_lastcommit stored procedure, passing it the *rs_origin*, *rs_origin_qid*, *-rs_secondary_qid*, and *rs_origin_commit_time* parameters.

```
create function string rs_dumptran
 for sqlserver_derived_class
 output language
 'execute dumptran_proc
     ?rs_dump_dbname!sys?,
     ?rs_dump_label!sys?,
     ?rs_dump_timestamp!sys?,
     ?rs_destination_db!sys?,
     ?rs_origin!sys?,
     ?rs_origin_qid!sys?,
     ?rs_secondary_qid!sys?
     ?rs_origin_commit_time!sys?'
```

If the server crashes after the dump is complete but before the rs_lastcommit system table is updated, Replication Server restarts the backup.

For an example of how dumptran_proc can be written, see the dumpdb_proc stored procedure example in rs_dumpdb.

Note There is no guarantee that the dump and the rs_update_lastcommit procedure will be executed atomically, because Adaptive Server does not allow the dump command to be included in a transaction with other commands. If the rs_lastcommit system table is not updated successfully, an additional dump may be performed.

Example 2 Alters the rs_dumptran function string that you created in the first example to execute as a remote procedure call.

```
alter function string rs_dumptran
 for sqlserver_derived_class
 output rpc
 'execute dumptran_proc
     ?rs_dump_dbname!sys?,
     ?rs_dump_label!sys?,
     ?rs_dump_timestamp!sy‘s?,
     ?rs_destination_db!sys?,
     ?rs_origin!sys?,
     ?rs_origin_qid!sys?,
     ?rs_secondary_qid!sys?,
     ?rs_origin_commit_time!sys?!'
```

Usage

- Replication Server coordinates transaction dumps by inserting an rs_dumptran function call at the same place in the stream of transactions it distributes to all replicate Replication Servers.
- rs_dumptran has function-string-class scope.

Note Replication Server does not initialize or generate rs_dumptran function strings for the system-provided function-string classes. You must create a function string before using a coordinated dump with Adaptive Server.

- Create an rs_dumptran function string at the Replication Server that is the primary site for the class.
- The rs_lastcommit system table should be updated when the rs_dumptran function string executes so that a restarted Replication Server does not perform duplicate dumps. See rs_commit for information about rs_lastcommit.
- To account for different dump devices at multiple replicate sites, create a stored procedure in each replicate database that performs a transaction dump, then write the rs_dumptran function string to execute the stored procedure.
- Table 4-3 lists the system variables used in rs_dumptran function strings.

Table 4-3: System variables for rs_dumptran function strings

Variable name	Datatype	Description
rs_dump_dbname	varchar(30)	The name of the database where the dump originated.
rs_dump_label	varchar(30)	Label information for the dump. For Adaptive Server, this variable contains a datetime value for the time the dump began.
rs_dump_timestamp	varbinary(16)	An Adaptive Server database timestamp taken when the dump was started at the origin. The variable is used for informational purposes only.

See also

create function string, rs_commit, rs_dumpdb, rs_get_lastcommit

rs_get_charset

Description

Returns the character set used by a data server. This function allows Replication Server to print a warning message if the character set is not what is expected.

Examples

Creates an rs_get_charset function string with output language that calls the sp_serverinfo system procedure and returns the data server's character set.

```
create function string rs_get_charset
 for rs_sqlserver2_function_class
 output language
 'sp_serverinfo server_csname'
```

Usage

- rs_get_charset obtains the name of the character set used by a data server. The Replication Server executes this function each time it connects to the data server.
- rs_get_charset has function-string class scope.
- Replication Server creates an initial rs_get_charset function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_get_charset function string.
- Create or customize an rs_get_charset function string at the Replication Server that is the primary site for the class.
- The default rs_get_charset function string for the rs_sqlserver_function_class and rs_default_function_class classes calls the Adaptive Server stored procedure sp_serverinfo with the argument *server_csname*.
- The data server should return a string with the name of a valid Sybase-supported character set. Valid Sybase character sets are defined in the Sybase release directory in *charsets/charset_name/charset.loc*, where each *charset_name* represents the name of a supported character set. For example, the file *charsets/iso_1/charset.loc* defines the iso_1 character set.

See also

create function string, rs_get_sortorder

rs_get_lastcommit

Description

Returns rows from the rs_lastcommit system table.

Examples

Creates an rs_get_lastcommit function string that executes a stored procedure named rs_get_lastcommit. The text of the stored procedure is:

```
create procedure rs_get_lastcommit
 as
 select origin, origin_qid, secondary_qid
   from rs_lastcommit
```

```
create function string rs_get_lastcommit
 for sqlserver_derived_class
 output language
 'execute rs_get_lastcommit'
```

Usage

- Replication Server executes rs_get_lastcommit when it starts up a DSI process for a database. The function returns all of the rows in the rs_lastcommit system table. Replication Server uses this information to find the last transaction committed from each primary data source.
- The rs_lastcommit system table is updated each time Replication Server commits a transaction in the database.
- rs_get_lastcommit has function-string-class scope.
- Replication Server creates an initial rs_get_lastcommit function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_get_lastcommit function string.
- Create or customize an rs_get_lastcommit function string at the Replication Server that is the primary site for the class.
- The default rs_get_lastcommit function string for the rs_sqlserver_function_class and rs_default_function_class classes updates the rs_lastcommit table by executing a stored procedure named rs_update_lastcommit in the rs_commit function string.
- rs_get_lastcommit must return columns in the correct order for each primary database whose data is replicated in the database. See Table 4-4.

Table 4-4: Columns returned by rs_get_lastcommit

Column name	Datatype	Description
origin	int	The ID number for the primary database the row represents

Column name	Datatype	Description
origin_qid	binary(36)	Identifies the last committed transaction in the stable queue for the origin database
secondary_qid	binary(36)	If a subscription materialization queue exists for the origin database, this column contains the last transaction in that queue that has been committed in the replicate database

See also create function string, rs_commit

rs_get_sortorder

Description

Obtains the sort order used by a data server. This function returns a warning message if the sort order does not match that of the Replication Server, and if the sort order is not what is expected.

Examples

Creates an rs_get_sortorder function string with output language that calls the sp_serverinfo system procedure and returns the data server's sort order.

```
create function string rs_get_sortorder
 for rs_sqlserver2_function_class
 output language
 'sp_serverinfo server_soname'
```

Usage

- The rs_get_sortorder function obtains the name of the sort order used by a data server. Replication Server executes this function each time it connects to the data server. If the sort order does not match that of the Replication Server, a warning message is written into the Replication Server error log. If the sort orders match, no warming message is written.
- The rs_get_sortorder function has function-string-class scope.
- Replication Server creates an initial rs_get_sortorder function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, you must create an rs_get_sortorder function string.
- If you need to create or customize an rs_get_sortorder function string, do so at the Replication Server that is the primary site for the class.
- The default rs_get_sortorder function string for the rs_sqlserver_function_class and rs_default_function_class classes calls the Adaptive Server stored procedure sp_serverinfo with the argument *server_soname*.
- An rs_get_sortorder function string should return a string with the name of a valid Sybase-supported sort order. Valid Sybase sort orders for a given character set are defined in the Sybase release directory in *charsets/charset_name/sortorder.srt*, where *charset_name* represents the name of a supported character set and *sortorder* represents the name of a supported sort order for the character set. For example, the file *charsets/iso_1/nocase.srt* defines the "nocase" sort order for the iso_1 character set.

See also

create function string, rs_get_charset

rs_get_textptr

Description

Retrieves the description for a text, unitext, or image column.

Examples

Creates an rs_get_textptr function string for the repcopy column in the blurbs table. The function string name, copy, is the name of the text, unitext, or image column in the replication definition.

```
create function string
 blurbs_rep.rs_get_textptr;copy
 for sqlserver2_function_class
 output language
 'select repcopy from blurbs
 where au_id = ?au_id!new?'
```

Usage

- Replication Server calls rs_get_textptr to retrieve a text, unitext, or image column description before it sends data with the Client-Library function ct_send_data.
- rs_get_textptr has replication definition scope.
- When you create a replication definition, Replication Server generates an rs_get_textptr function string for the rs_sqlserver_function_class and rs_default_function_class classes for each replicated text, unitext, or image column in the replication definition.
- If you use a user-created base function-string class, you must create an rs_get_textptr function string for each replicated text, unitext, or image column included in the replication definition.
- Create or customize an rs_get_textptr function string at the Replication Server where you created the replication definition.
- rs_get_textptr must return a text or unitext column description for a text, unitext, or image column in a specified row. The text or unitext column description must conform to Open Server requirements for returning an “I/O descriptor structure.” For information about this structure, refer to the *Open Server Server-Library/C Reference Manual*.

See also

rs_datarow_for_writetext, rs_textptr_init, rs_writetext

rs_get_thread_seq

Description	Returns the sequence number for the specified entry in the rs_threads system table.
Syntax	rs_get_thread_seq @*rs_id*
Parameters	*rs_id* a number of int datatype. It represents the ID of the entry to be checked and matches the value of the id column in the rs_threads system table.

Examples

Creates an rs_get_thread_seq function string that executes a select statement in the rs_threads table.

```
create function string rs_get_thread_seq
 for sqlserver_derived_class
 output language
 'select seq from rs_threads
   where id = ?rs_id!param?'
```

Usage

- Replication Server executes rs_get_thread_seq to check the completion status of preceding transactions. It is executed only when more than one DSI thread is defined for a connection. The function returns a single row with a single column, seq, which contains the sequence number for the specified ID.
- The thread invoking this function is blocked until the transaction that last modified the specified entry completes its transaction.
- rs_get_thread_seq has function-string-class scope.
- Replication Server creates an initial rs_get_thread_seq function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class and you use the parallel DSI feature, you must create a function string for the rs_get_thread_seq function. If you do not use parallel DSI, you do not need to create a function string for this function.
- Create or customize an rs_get_thread_seq function string at the Replication Server that is the primary site for the class.

See also

configure connection, rs_initialize_threads, rs_set_isolation_level, rs_update_threads

rs_get_thread_seq_noholdlock

Description

Returns the sequence number for the specified entry in the rs_threads system table, using the noholdlock option.

Syntax

rs_get_thread_seq_noholdlock @*rs_id*

Parameters

rs_id

a number of int datatype. It represents the ID of the entry to be checked and matches the value of the id column in the rs_threads system table.

Examples

Creates an rs_get_thread_seq_noholdlock function string that executes a select statement on the rs_threads table.

```
create function string
   rs_get_thread_seq_noholdlock
 for sqlserver_derived_class
 output language
 'select seq from rs_threads noholdlock
   where id = ?rs_id!param?'
```

Usage

- rs_get_thread_seq_noholdlock is equivalent to rs_get_thread_seq, except that it is used when dsi_isolation_level is 3. It is executed only when more than one DSI thread is defined for a connection. The row select is done with the noholdlock option. The function returns a single row with a single column, seq, which contains the current sequence number for the specified ID.
- The rs_get_thread_seq_noholdlock function has function-string class scope.
- Replication Server creates an initial rs_get_thread_seq_noholdlock function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class and you use the parallel DSI feature with transaction isolation level 3, create a function string for rs_get_thread_seq_noholdlock.
- Create or customize an rs_get_thread_seq_noholdlock function string at the Replication Server that is the primary site for the class.

See also

alter connection, rs_get_thread_seq, rs_initialize_threads, rs_set_isolation_level, rs_update_threads

rs_initialize_threads

Description

Sets the sequence of each entry in the rs_threads system table to 0.

Syntax

rs_initialize_threads *@rs_id*

Parameters

@rs_id
a number from 1 through *dsi_num_threads*, representing the ID of the entry Replication Server will set to 0.

Examples

Creates an rs_initialize_threads function string that executes a stored procedure named rs_initialize_threads. The text of the stored procedure is:

```
create procedure rs_initialize_threads
   @rs_id int
 as
     delete from rs_threads where id = @rs_id
     insert into rs_threads values
       (@rs_id, 0,"", "", "", "")
```

```
create function string rs_initialize_threads
 for sqlserver_derived_class
 output language
 'execute rs_initialize_threads
   @rs_id = ?rs_id!param?'
```

Usage

- rs_initialize_threads Replication Server executes function when a connection is initialized. It is executed only when more than one DSI thread is defined for the connection. It sets the sequence number of each entry in the rs_threads system table to 0.
- rs_initialize_threads has function-string-class scope.
- Replication Server creates an initial rs_initialize_threads function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class and you use the parallel DSI feature, create a function string for rs_initialize_threads.
- Create or customize an rs_initialize_threads function string at the Replication Server that is the primary site for the class.

See also

create connection, rs_get_thread_seq, rs_get_thread_seq_noholdlock, rs_set_isolation_level, rs_update_threads

rs_insert

Description

Inserts a single row into a table in a replicate database.

Examples

Replaces the rs_insert function string for the publishers table.

```
alter function string publishers.rs_insert
 for sqlserver_derived_class
 output language
 'insert into publishers (pub_id, pub_name, city,
  state)
 values (?pub_id!new?, ?pub_name!new?,
     ?city!new?, ?state!new?)'
```

Usage

- rs_insert has replication definition scope.
- Replication Server generates an rs_insert function string for the system-provided function-string classes when you create a replication definition.
- If you use a user-created base function-string class, create an rs_insert function string for each replication definition.
- Create or customize an rs_insert function string at the Replication Server where you created the replication definition.
- The default generated function string for rs_insert, for the rs_sqlserver_function_class and rs_default_function_class classes for each replication definition, uses the Transact-SQL insert command syntax.
- Replication Server cannot send text, unitext, or image data to a replicate database in rs_insert, but it can report the status of text, unitext, or image data with the *text_status* modifier. For a description of the *text_status* modifier, see rs_datarow_for_writetext. text, unitext, or image data is sent to the replicate database with rs_get_textptr, rs_textptr_init, and rs_writetext.

See also

create function string, create replication definition, rs_datarow_for_writetext, rs_delete, rs_get_textptr, rs_select, rs_select_with_lock, rs_textptr_init, rs_update,

rs_marker

Description	Passes its parameter to Replication Server as an independent command.
Syntax	rs_marker @*rs_api*

Parameters

rs_api
a varchar(255) character string that contains data used for subscription materialization.

Examples

```
create function string rs_marker
 for sqlserver_derived_class
 output language
 'execute rs_marker
     @rs_api = ?rs_api!param?'
```

Usage

- rs_marker allows Replication Server to insert data into the transaction log so that it can be retrieved by the RepAgent thread.
- The rs_marker function has function-string-class scope.
- Replication Server creates an initial rs_marker function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, create a function string for the rs_marker function.
- Create or customize an rs_marker function string at the Replication Server that is the primary site for the class.
- Replication Server uses rs_marker during subscription materialization to pass the activate subscription and validate subscription commands to the primary Replication Server via the primary database log.
- The RepAgent for the primary database must recognize an rs_marker function execution and pass the *@rs_api* parameter to the primary Replication Server as a command.
- For Adaptive Server databases, an Adaptive Server replicated stored procedure named rs_marker is created when the database is set up for Replication Server. This stored procedure is marked “replicated” using the sp_setrepproc system procedure.

- When the Adaptive Server RepAgent encounters an rs_marker execution in the transaction log, it sends the *@rs_api* parameter to the primary Replication Server as a command.

Note Do not change the rs_marker function string or invoke the rs_marker stored procedure except when you create bulk subscriptions as described in the *Replication Server Administration Guide Volume 1*.

See also activate subscription, create subscription, sp_setrepproc, validate subscription

rs_raw_object_serialization

Description

Enables Replication Server to process Java columns in serialized format.

Usage

- rs_raw_object_serialization allows Replication Server to insert serialized data directly into the replicate database.
- rs_raw_object_serialization has function-string class scope.
- Replication Server creates an initial rs_raw_object_serialization function string for the system-provided function-string classes rs_sql-server_function_class and rs_default_function_class during installation.
- Replication Server uses rs_raw_object_serialization when the first Java column is materialized or replicated for a connection, passing the default command set rs_raw_object_serialization on to the Adaptive Server.

rs_repl_off

Description

Specifies whether transactions executed by the maintenance user in the Adaptive Server database are replicated.

Examples

Creates an instance of an rs_repl_off function string.

```
create function string rs_repl_off
 for sqlserver_derived_class
 output language
 'set replication off'
```

Usage

- rs_repl_off is executed for the DSI connection to a standby database.
- rs_repl_off has function-string-class scope.
- Replication Server creates an initial rs_repl_off function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, create a function string for rs_repl_off if you plan to use it in any way other than the default.
- Create or customize an rs_repl_off function string at the Replication Server that is the primary site for the class.
- Standby database connections always use the system-provided class rs_default_function_class, which cannot be modified. Therefore, if you are not using warm standby, you do not need to create a function string for rs_repl_off.
- You can use alter connection or configure connection to set the dsi_replication configuration parameter and to specify whether or not to execute the rs_repl_off function when connecting to the standby database. Set dsi_replication to “off” to execute rs_repl_off.
- In a warm standby application, Replication Server sets dsi_replication to “on” for the active database and to “off” for the standby database.

See also

create connection, create function string

rs_repl_on

Description

Sets replication on in Adaptive Server for either a database connection or database connections.

Examples

Creates an instance of an rs_repl_on function string:

```
create function string rs_repl_on
 for sqlserver_derived_class
 output language
 'set replication on'
```

Usage

- rs_repl_on is executed for the DSI connection to a database.
- rs_repl_on has function-string class scope.
- Replication Server creates an initial rs_repl_on function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, create a function string for rs_repl_on if you plan to use it in any way other than the default.
- Create or customize an rs_repl_on function string at the Replication Server that is the primary site for the class.

See also

alter connection, rs_repl_off

rs_rollback

Description

Rolls back a transaction. This function is reserved for future use.

Examples

This example illustrates the default rs_rollback function string for the rs_sqlserver_function_class and rs_default_function_class classes.

```
create function string rs_rollback
 for sqlserver_derived_class
 output language
 'rollback transaction'
```

Usage

- Rolled back transactions retrieved from a primary database transaction log are not distributed to replicate Replication Servers, so this function should never be executed.
- The rs_rollback function has function-string-class scope.
- Replication Server creates an initial rs_rollback function string for the system-provided function-string classes during installation.

See also

alter function string, create function string, rs_begin, rs_commit

rs_select

Description

Selects rows for subscription materialization from the primary copy of a replicated table and, for subscription dematerialization, from the replicate copy of the table.

Examples

Creates an instance of an rs_select function string. Replication Server uses this function string when a subscription where clause specifies a specific value for the au_lname column.

```
create function string
   authors.rs_select;name_select
 for flat_file_class
 scan 'select * from authors
   where au_lname = ?l_name!user?'
 output rpc
 'execute name_sel ?l_name!user?, "authors"'
```

Usage

- Replication Server executes rs_select to retrieve subscription materialization rows from the primary Replication Server when without holdlock is included in the create subscription command. without holdlock is used in non-atomic materialization. The function string used for this operation is in the class assigned to the primary database.
- To retrieve data during atomic materialization, use the function-string class and error class associated with the primary database connection, not the classes associated with the replicate database connection.
- Replication Server also executes rs_select to identify rows for subscription dematerialization, if you drop a subscription for a table replication definition using incrementally with purge. The function string used for this operation is in the class assigned to the replicate database.
- If create subscription does not include without holdlock, Replication Server executes the rs_select_with_lock function instead of rs_select.
- rs_select has replication definition scope.
- Replication Server generates rs_select function strings for the system-provided function-string classes when you create a replication definition.
- If you use a user-created base function-string class, create rs_select function strings for each replication definition to match each possible subscription where clause.
- Create or customize an rs_select function string at the Replication Server where you created the replication definition.

- The default generated function strings for rs_select, for the rs_sqlserver_function_class and rs_default_function_class classes for each replication definition, use the Transact-SQL select command syntax.
- Function strings for rs_select have input and output templates. The input template is a SQL select command with a where clause that Replication Server matches with the where clause in the create subscription command.
- If Replication Server cannot match the where clause in a select operation to a function string input template, it uses a function string with no input template, if one exists.
- An rs_select function call fails if Replication Server cannot locate a function string with a matching input template or a function string with no input template.

See also

alter function string, create function string, create subscription, rs_delete, rs_insert, rs_select_with_lock, rs_update

rs_select_with_lock

Description

Selects rows for subscription materialization from the primary copy of a replicated table, using a holdlock to maintain serial consistency.

Examples

Creates an instance of an rs_select_with_lock function string. Replication Server uses this function string when a subscription where clause specifies a value for the au_lname column.

```
create function string
   authors.rs_select_with_lock;name_select
 for flat_file_class
 scan 'select * from authors
   where au_lname = ?l_name!user?'
 output rpc
 'execute name_sel_lock ?l_name!user?, "authors"'
```

Usage

- Replication Server executes the rs_select_with_lock function to retrieve initial subscription rows from the primary Replication Server when the without holdlock clause is used with create subscription. The without holdlock clause is not used in atomic materialization. The function string used for this operation is in the class assigned to the primary database.
- Replication Server also executes rs_select_with_lock to identify rows for subscription dematerialization if you drop a subscription for a table replication definition using with purge. The function string used for this operation is in the class assigned to the replicate database.
- If the without holdlock clause is included in create subscription, Replication Server executes the rs_select function instead of rs_select_with_lock.
- rs_select_with_lock has replication definition scope.
- Replication Server generates rs_select_with_lock function strings for the system-provided function-string classes when you create a replication definition.
- If you use a user-created base function-string class, create an rs_select_with_lock function string for each replication definition to match each possible subscription where clause.
- Create or customize an rs_select_with_lock function string at the Replication Server where you created the replication definition.
- The default generated function strings for rs_select_with_lock, for the rs_sqlserver_function_class and rs_default_function_class classes for each replication definition, use the Transact-SQL select...holdlock command syntax.

- Function strings for rs_select_with_lock have input and output templates. The input template is a SQL select command with a where clause that Replication Server matches with the where clause in the create subscription command.
- If Replication Server cannot match the where clause in a select operation to a function-string input template, it uses a function string with no input template, if one exists.
- An rs_select_with_lock function call fails if Replication Server cannot locate a function string with a matching input template or a function string with no input template.

See also

alter function string, create function string, create subscription, rs_delete, rs_insert, rs_select, rs_update

rs_set_ciphertext

Description

Enables replication of encrypted columns to an Adaptive Server table.

Examples

Alters rs_set_ciphertext for non-Adaptive Server databases that do not support “set ciphertext on”:

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

Usage

- rs_set_ciphertext is called after rs_usedb for any user database connection. Replication Server does not call this function string for Replication Server connections and RSSD connections.
- rs_set_ciphertext issues "set ciphertext on" for the rs_default_function_class and the rs_sqlserver_function_class. For all other function classes, rs_set_ciphertext is set to null (an empty string).
- In case of failure, Replication Server continues running and does not report back to the user. This is for backward compatibility with older versions of Adaptive Server that do not support “set ciphertext on”.
- Encrypted columns come to Replication Server in varbinary, encrypted form. For materialization and dematerialization, Replication Server must either “set ciphertext on” for the database connection, or call the Adaptive Server ciphertext() function.

- Replication Server always sets the ciphertext property on, whether there is an encrypted column to be replicated, or whether the target database accepts ciphertext property.
- Do not specify encrypted columns as searchable. Replication Server does not know if a varbinary column is ciphertext or plain binary and cannot prevent an encrypted column being a search column.
- Do not map encrypted columns to other than varbinary datatypes. Replication Server does not know if a column is encrypted or not and cannot prevent ciphertext being converted to other datatypes.
- Replication Server cannot encrypt text, unitext, and image columns.

See also

alter connection, alter function string, create database replication definition, create replication definition

rs_set_dml_on_computed

Description

Enables the replication of materialized computed columns to the replicate Adaptive Server database as regular columns.

Usage

- rs_set_dml_on_computed maps to the command set dml_on_computed “on” for Adaptive Server replicate databases. For all non-Sybase databases, this function maps to null.
- rs_set_dml_on_computed has function-string class scope.
- rs_set_dml_on_computed is always applied at DSI after the use database command when connection is established.
- set dml_on_computed “on” is not supported by Adaptive Server version 12.5.*x* and earlier databases. In case of failure, Replication Server will continue running and will not report back to user.

See also

create replication definition

rs_set_isolation_level

Description

Passes the isolation level for transactions to the replicate data server.

Examples

Creates an instance of an rs_set_isolation_level function string.

```
create function string rs_set_isolation_level
for sqlserver_derived_class
output language
'set transaction isolation level?rs_isolation_level!sys_raw?'
```

Usage

- The rs_set_isolation_level function passes the transaction isolation level to the replicate data server, and executes every time the DSI connects to the replicate data server if a value has been set for dsi_isolation_level. If the dsi_isolation_level is the default value, rs_set_isolation_level is not executed.
- Use the alter connection or create connection with the set_isolation_level option to the value for the variable *rs_isolation_level*. The supported Adaptive Server values are 0, 1, 2, and 3. Replication Server supports all other isolation level values supported by other data servers. If no value is supplied for *rs_isolation_level*, Replication Server uses the isolation value of the target data server.
- Replication Server executes rs_set_isolation_level immediately after executing the rs_usedb function-string command.
- The rs_set_isolation_level function has function-string class scope.
- Replication Server creates an initial rs_set_isolation_level function string for the Adaptive Server and default function-string classes during installation.
- If you use a nondefault function-string class and you use the parallel DSI feature, create a function string for the rs_set_isolation_level function. The modified function string must contain the variable *rs_isolation_level*.
- Create or customize an rs_set_isolation_level function string at the Replication Server that is the primary site for the class.

See also

create connection, rs_get_thread_seq, rs_initialize_threads, rs_update_threads

rs_setproxy

Description

Changes the login name in a data server.

Usage

- rs_setproxy has function-string-class scope.
- Replication Server creates an rs_setproxy function string for the rs_sqlserver_function_class function-string class during installation. The default value is:

  ```
  set session authorization "?rs_destination_user!sys"
  ```

  The generated string has the syntax of the Adaptive Server set proxy command. Use alter function string to replace the default function string.
- If a data server does not support network security services or does not have a corresponding set proxy command, you can either turn unified_login to “not required” or create an empty rs_setproxy function string.
- Function-string variable modifiers *sys* contains the login name of a data server. This login name is usually that of the maintenance user or the subscription user.

See also

alter function string, create function string

rs_textptr_init

Description

Allocates a text pointer for a text, unitext, or image column.

Examples

Creates an rs_textptr_init function string for the copy column in the blurbs table.

```
create function string
   blurbs_rep.rs_textptr_init;copy
 for sqlserver2_function_class
 output language
 'update blurbs set copy = NULL
 where au_id = ?au_id!new?'
```

Usage

- Replication Server executes rs_textptr_init when a row arrives, indicating that modifications were made at the primary database, which caused a text pointer allocation for the text, unitext, or image column. It it also executed when Replication Server needs to do a writetext operation at the replicate database and the text pointer has not been allocated.
- The rs_textptr_init function has replication definition scope.
- For each replicated text, unitext, or image column in a replication definition, Replication Server generates an rs_textptr_init function string for the rs_sqlserver_function_class and rs_default_function_class classes when you create the replication definition.
- If you use a user-created base function-string class, create an rs_textptr_init function string for each replicated text, unitext, or image column included in the replication definition.
- Create or customize an rs_textptr_init function string at the Replication Server where you created the replication definition.

See also

rs_get_textptr, rs_datarow_for_writetext, rs_writetext

rs_ticket_report

Description
Invokes the replicate database stored procedure rs_ticket_report.

Examples
A sample of the customized rs_ticket_report:

```
Create procedure rs_ticket_report
  @rs_ticket_param varchar(225)
  Begin
   set nocount on

   declare @t_new  varchar(225),
   @c_time datetime

   -- For a string: "@rs_ticket_param;RDB(name)=hh:mm:ss.ddd"
   select @c_time = getdate()
   select @t_new = @rs_ticket_param + ";RDB(" + db_name() + ")="
          + convert(varchar(8), @c_time, 8) + "." + right("00"
          + convert(varchar(3), datepart(ms, @c_time)), 3)
   insert rs_ticket_history values (@c_time, @t_new)
end
```

Usage

- rs_ticket_report has function-string class scope.
- rs_ticket_report has one parameter called *rs_ticket_param*, which contains timestamp and byte information for EXEC, DIST, and DSI transactions.
- To use the rs_ticket_report function string, set the connection configuration parameter dsi_rs_ticket_report to on.
- By default, the rs_ticket_report function string invokes the replicate database stored procedure rs_ticket_report. You can customize and use this stored procedure to alter rs_ticket_report function string to invoke different stored procedures.

See also
rs_ticket, rs_ticket_report stored procedure

rs_triggers_reset

Description

Turns off triggers in the Adaptive Server.

Examples

Creates an instance of an rs_triggers_reset function string for a user-created base function-string class.

```
create function string rs_triggers_reset
 for sqlserver2_function_class
 output language
 'set triggers off'
```

Usage

- By default, the rs_triggers_reset function is executed for the DSI connection to a standby database, and is not executed for any other DSI connection.
- rs_triggers_reset has function-string-class scope.
- During installation, Replication Server creates an initial rs_triggers_reset function string for the system-provided function-string classes.
- Standby database connections always use the system-provided class rs_default_function_class, which cannot be modified. For any other database connection, you do not need to create a function string for the rs_triggers_reset function, unless:
    - The database connection uses a user-created base function-string class, and
    - You want to set the dsi_keep_triggers configuration parameter to “off” for the connection.
- Create an rs_triggers_reset function string at the Replication Server that is the primary site for the class.
- Setting dsi_keep_triggers to “off” for a database connection to execute rs_triggers_reset when the connection is established. The dsi_keep_triggers default is “off” for standby databases, and “on” for replicate databases. Use the alter connection or configure connection command to change this setting.

See also

create connection, create function string

rs_truncate

Description

Truncates a table or a table partition in a replicate database.

Examples

Example 1 Replaces the existing rs_truncate function string for the authors table with one that executes a Transact-SQL delete command, which logs all deletions, instead of the truncate table command, which does not log deletions.

```
alter function string authors.rs_truncate
 for sqlserver_derived_class
 output language
 'delete authors'
```

You would want to customize the rs_truncate function string for the authors table, if:

- The replicate database doesn't support table the Transact-SQL truncate table command, or
- You want to have deletions logged at the replicate database.

Example 2 Replaces the existing rs_truncate function string for the publisher table to replicate truncate table partition as a delete command:

```
alter function string publisher.rs_truncate
  for rs_sqlserver_function_class
  output language
  'begin transaction
    if (?1!param? = '''') /* No parameter */
      delete publisher
    if (?1!param? = ''A'')
      delete publisher where c1 < 1000
    if (?1!param? = ''B'')
      delete publisher where c1 >= 1000
commit transaction'
```

Example 3 Alters the function string to do nothing if there is a parameter so that table partitions are not truncated at replicate:

```
alter function string publisher.rs_truncate
  for rs_sqlserver_function_class
  output language
  'if(?1!param? = '''') delete publisher'
```

Usage

- rs_truncate has a replication definition scope. Replication Server executes it to truncate a table or one or more table partitions.

- Replication Server generates an rs_truncate function string for the system-provided function-string classes when you create the replication definition.
- If you use a user-created base function-string class, create an rs_truncate function string for each replication definition.
- Create or customize an rs_truncate function string at the Replication Server where you created the replication definition.
- The default-generated function string for rs_truncate, for the rs_sqlserver_function_class and rs_default_function_class classes for each replication definition, uses the Transact-SQL truncate table command syntax. It deletes all rows in a table without logging the deletion of each individual row.
- Replication Server will reconstruct the same command executed at the primary. This command requires that the replicate site to have the same partition names. If not, DSI will shut down.
- The partition names are passed as parameters to the rs_truncate function. rs_truncate function string accepts position-based function-string parameters. The following is a position-based variable:

  ```
  ?n!param?
  ```

  The function-string variable ?1!param? corresponds to the first parameter in the rs_truncate function.
- Table 4-5 lists the function string variable modifiers.

Table 4-5: Function string variable modifiers

Modifier	Description
new, new_raw	A reference to the new value of a column in a row you are inserting or updating
old, old_raw	A reference to the existing value of a column in a row you are updating or deleting
user, user_raw	A reference to a variable that is defined in the input template of an rs_select or rs_select_with_lock function string
sys, sys_raw	A reference to a system-defined variable
param, param_raw	A reference to a function parameter
text_status	A reference to or a function parameter. If the parameter is not defined through function replication definition (create function replication definition) or user defined function (create function), there must be a number between 1 and 99 (with no leading 0) in place of parameter name which states the parameter position in the function in the LTL command.

- A function string has a minimum version of 1500 if it contains position-based function-string variables. A replication definition has a minimum version of at least 1500 if it contains a 1500 function string.

See also

alter function string, rs_datarow_for_writetext, rs_get_textptr, rs_insert, rs_delete, rs_textptr_init, rs_writetext, set autocorrection

rs_update

Description

Updates a single row in a table in a replicate database.

Examples

Replaces the existing rs_update function string for the authors table with one that is similar to the default function string generated by Replication Server for the system-provided function-string classes.

```
alter function string authors.rs_update
 for sqlserver_derived_class
 output language
 'update authors set au_id = ?au_id!new?,
     au_lname = ?au_lname!new?,
     au_fname = ?au_fname!new?,
     phone = ?phone!new?,
     address = ?address!new?,
     city = ?city!new?,
     state = ?state!new?,
     country = ?country!new?,
     postalcode = ?postalcode!new?
   where au_id = ?au_id!old?'
```

Usage

- Replication Server executes rs_update to update a single row in a table. The row is identified by the primary key columns defined in a replication definition for the table.
- The rs_update function has replication definition scope.
- Replication Server generates an rs_update function string for the system-provided function-string classes when you create the replication definition.
- If you use a user-created base function-string class, create an rs_update function string for each replication definition.
- Create or customize an rs_update function string at the Replication Server where you created the replication definition.
- The default generated function string for rs_update, for the rs_sqlserver_function_class and rs_default_function_class classes for each replication definition, uses the Transact-SQL update command syntax. It replaces all columns in the row, and identifies the row with a where clause that specifies the pre-update values, or before image, of the primary key columns.
- When set autocorrection is on, Replication Server does not use rs_update. Instead, it calls rs_delete to remove the existing row and rs_insert to insert the row.

- Replication Server cannot send text, unitext, or image data with rs_update, but it can report the status of text, unitext, or image data with the *text_status* modifier. For a description of the *text_status* modifier, see rs_datarow_for_writetext. Data of type text, unitext, or image is sent to the replicate database with the rs_get_textptr, rs_textptr_init, rs_datarow_for_writetext, and rs_writetext functions.

See also

alter function string, rs_datarow_for_writetext, rs_get_textptr, rs_insert, rs_delete, rs_textptr_init, rs_writetext, set autocorrection

rs_update_threads

Description

Updates the sequence number for the specified entry in the rs_threads system table.

Syntax

rs_update_threads @*rs_id*, @*rs_seq*

Parameters

rs_id
a number of int datatype representing the ID of the entry to be updated.

rs_seq
a number of int datatype representing the new sequence number for the entry.

Examples

Creates an rs_update_threads function string that executes a stored procedure named rs_update_threads. The text of the stored procedure is:

```
create function string rs_update_threads
  for sqlserver_derived_class
  output language
  'execute rs_update_threads
  @rs_seq = ?rs_seq!param?,
  @rs_id = ?rs_id!param?'
```

```
create procedure rs_update_threads
  @rs_id int,
  @rs_seq int
  as
  update rs_threads set seq = @rs_seq
    where id = @rs_id
```

Usage

- The rs_update_threads function is executed at the start of each transaction when more than one DSI thread is defined for a connection. It is executed only when more than one DSI thread is defined for a connection.
- The rs_update_threads function has function-string-class scope.
- Replication Server creates an initial rs_update_threads function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class and the parallel DSI feature, create a function string for rs_update_threads.
- Create or customize an rs_update_threads function string at the Replication Server that is the primary site for the class.

See also

create connection, rs_get_thread_seq, rs_initialize_threads, rs_set_isolation_level

rs_usedb

Description

Changes the database context in a data server.

Examples

Example 1 Changes an existing rs_usedb function string to one that is similar to the default function string generated by Replication Server for the system-provided function-string classes.

```
alter function string rs_usedb
 for sqlserver_derived_class
 output language
 'use ?rs_destination_db!sys_raw?'
```

Example 2 Creates an rs_usedb function string with an empty string for an output template for a data server that does not support multiple databases.

```
create function string rs_usedb
 for TOKYO_DS
 output language ''
```

Usage

- The Replication Server DSI executes the function when it first connects to the data server.
- rs_usedb has function-string class scope.
- Replication Server creates an initial rs_usedb function string for the system-provided function-string classes during installation.
- If you use a user-created base function-string class, create a function string for the rs_usedb function.
- Create or customize an rs_usedb function string at the Replication Server that is the primary site for the class.
- The default generated function string for the rs_usedb function, for the rs_sqlserver_function_class and rs_default_function_class classes, has the syntax of the Transact-SQL use command.
- If a data server does not support multiple databases or a database context, the output template can be an empty string (' ').

See also

alter function string, create function string

rs_writetext

Description

Modifies text, unitext, or image data in a replicate database.

Examples

Example 1 Creates an rs_writetext function string that uses the RPC method to update the copy column in the blurbs table.

```
create function string
   blurbs_rep.rs_writetext;copy
 for gw_function_class
 output rpc
 'execute update_blurbs_copy
   @copy_chunk = ?copy!new?,
   @au_id = ?au_id!new?,
   @last_chunk = ?rs_last_text_chunk!sys?,
   @writetext_log = ?rs_writetext_log!sys?'
```

Example 2 Creates an rs_writetext function string that uses the writetext method to update the copy column. Replication Server modifies the copy column by using the I/O descriptor returned by the execution of the rs_get_textptr function for the copy column.

```
create function string
   blurbs_rep.rs_writetext;copy
 for rs_sqlserver2_function_class
 output writetext
 use primary log
```

For example, if you have a function string for rs_get_textptr, then the rs_writetext function modifies the repcopy column in the blurbs table, as follows:

```
create function string
   blurbs_rep.rs_get_textptr;copy
 for sqlserver2_function_class
 output language
 'select repcopy from blurbs
 where au_id = ?au_id!new?'
```

Example 3 Creates an rs_writetext function string that uses the none method to specify that the copy column should not be updated.

```
create function string
   blurbs_rep.rs_writetext;copy
 for rs_sqlserver2_function_class
 output none
```

Usage

- rs_writetext has replication definition scope.

- For each replicated text, unitext, or image column in a replication definition, Replication Server generates an rs_writetext function string for the rs_sqlserver_function_class and rs_default_function_class classes when you create the replication definition.
- If you use a user-created function-string class, create an rs_writetext function string for each replicated text, unitext, or image column included in the replication definition.
- Create or customize an rs_writetext function string at the Replication Server where you created the replication definition.
- Replication Server supports three output formats for creating an rs_writetext function string: RPC, writetext, and none.

Using the RPC Method

With the RPC method for creating an rs_writetext function string, Replication Server executes a remote procedure call repeatedly, providing up to 255 bytes of the text, unitext, or image value on each procedure execution.

The data is passed in the RPC in a varchar parameter for text or unitext data or in a varbinary parameter for image data. Replication Server ensures that the data chunks are partitioned on character boundaries for text or unitext columns. If a 1-byte character set is in use, the data is sent in 255-byte chunks.

Each time Replication Server executes the RPC, it sets the *rs_last_text_chunk* system variable, an int, to 0 if there is more data to follow or to 1 if this is the last RPC execution for this text column.

- Another int system variable, *rs_writetext_log*, is set to 1 if the writetext logging option was used in the primary database or 0 if the logging option was not used in the primary database.
- The values of other columns in the data row can be accessed by using the *new* or *old* modifier. If you used the Transact-SQL insert command at the primary database, you must use the *new* modifier.
- Use the *text_status* modifier to retrieve the status of a text, unitext, or image column. For a description of the *text_status* modifier, see rs_datarow_for_writetext.

Using the *writetext* method

The writetext method for creating an rs_writetext function string provides the options shown in Table 4-6 to specify the logging behavior in the replicate database.

Table 4-6: writetext logging options

Logging option	Description
use primary log	Log the data in the replicate database transaction log if the logging option was specified in the primary database transaction log. Do not log if logging is not specified in the primary database transaction log.
with log	Log the data in the replicate database transaction log.
no log	Do not log the data in the replicate database transaction log.

The default function string for rs_sqlserver_function_class uses the use primary log option.

Using the *none* method

The none output template option for rs_writetext function strings instructs Replication Server not to use the Client-Library function ct_send_data to update a text, unitext, or image column value. This option provides necessary flexibility for using text, unitext, or image columns in a heterogeneous environment.

See the *Replication Server Administration Guide Volume 2* for more information.

See also

rs_get_textptr, rs_textptr_init, rs_datarow_for_writetext

CHAPTER 5 Adaptive Server Commands and System Procedures

This chapter contains reference pages for the Adaptive Server commands and system procedures used with Replication Server.

dbcc dbrepair

Description	A Transact-SQL command that clears the secondary truncation point for an offline replicated database.
Syntax	dbcc dbrepair(*database_name*, ltmignore)

Parameters

database_name
The name of the database for which you want to clear the secondary truncation point.

ltmignore
Deactivates the secondary truncation point in the named database.

Usage

- dbcc dbrepair clears the secondary truncation point for offline databases; dbcc settrunc with the ignore option clears the secondary trunction point for online databases.
- Sybase recommends that you drain the transaction log and clear the secondary truncation point for a replicated database before starting an upgrade. If you have not performed these two tasks, Adaptive Server does not allow you to bring the database online after upgrade.
- If you do not drain the transaction log and clear the secondary truncation point before upgrade, use dbcc dbrepair so that Adaptive Server can bring the database online.

  Before running dbcc dbrepair:

  a Start the RepAgent thread on the offline database.

  b Drain the transaction log.

  If you do not drain the transaction log before running dbcc dbrepair, all transactions in the log are lost.

See also dbcc settrunc

dbcc gettrunc

Description

A Transact-SQL command to retrieve current RepAgent information about an Adaptive Server database.

Syntax

dbcc gettrunc

The dbcc gettrunc command returns a single row containing the columns shown in Table 5-1:

Table 5-1: Columns returned by dbcc gettrunc

Column name RepAgent	Contents
secondary trunc page	The first page that is not truncated in the database log
secondary trunc state	One of the following values: • 1 – Adaptive Server does not truncate the log on or after the truncpage • 0 – Adaptive Server ignores the truncpage
db rep stat	A mask constructed of the following: • 0x01 – truncpage is valid • 0x02 – database contains replicated tables • 0x04 – database contains replicated stored procedures • 0x10 – replicate all to standby database RepAgent only: • 0x20 – RepAgent enabled • 0x40 – autostart RepAgent
generation id	The database generation ID
database id	The Adaptive Server ID number of the database
database name	The name of the database
ltl version	RepAgent: The log transfer language (LTL) version

Usage

Use dbcc gettrunc for RepAgent-enabled databases.

See also

admin get_generation, dbcc settrunc

dbcc settrunc

Description

A Transact-SQL command that modifies the secondary truncation point information for an Adaptive Server database.

Syntax

```
dbcc settrunc('ltm', {'valid' | 'ignore'})
dbcc settrunc('ltm', 'gen_id', db_generation)
```

Parameters

valid
: Instructs Adaptive Server to respect the secondary truncation point. This option prevents the Adaptive Server from truncating transaction log records that have not been transferred to Replication Server.

ignore
: Instructs Adaptive Server to ignore the secondary truncation point. This allows Adaptive Server to truncate log records that the RepAgent has not yet transferred to the Replication Server.

gen_id
: Instructs Adaptive Server to reset the database generation number in the log.

db_generation
: The new database generation number. Increment the number after restoring dumps to prevent Replication Server from rejecting new transactions as duplicates.

Warning! You cannot execute dbcc settrunc when RepAgent is running.

Usage

- Use dbcc settrunc for RepAgent-enabled databases.
- The secondary truncation point must be valid for Adaptive Server databases containing primary data to be replicated or for databases where replicated stored procedures are stored.
- When the secondary truncation point is valid, Adaptive Server does not truncate log records that the Replication Server has not yet received from the RepAgent.

- If the secondary truncation point is not modified for an extended period of time, the log may fill up and prevent applications from continuing. You can change the secondary truncation point to ignore—after shutting down the Replication Server and the RepAgent—so that the log can be truncated and applications can continue working. Then use the rs_zeroltm procedure to reset the locator value to zero (0). However, note this warning:

  Warning! If you set the secondary truncation point to ignore and then truncate the log, replicated data will be incorrect. You must either re-create subscriptions, reconcile subscriptions by executing rs_subcmp, or load database and transaction dumps and replay the lost transactions. See the *Replication Server Administration Guide Volume 2* for instructions for replaying lost transactions.You should increment the database generation number after restoring coordinated dumps. Use admin get_generation to find the current generation number.

  See rs_zeroltm on page 497 for details about running this stored procedure.

- Increment the database generation number after restoring to prevent Replication Server from rejecting new log records. See the *Replication Server Administration Guide Volume 2* for information about reloading coordinated dumps.

- If the primary Replication Server is unable to accept transactions and the primary database transaction log is full and must be truncated, you may need to turn off the secondary truncation point and truncate the log in order to allow Adaptive Server transactions to continue. In this situation, use dbcc settrunc('ltm', 'ignore') to shut down the Replication Agent and turn off the secondary truncation point in the database.

  After using dbcc settrunc, you *must* use the rs_zeroltm stored procedure to reset the locator value for a database to 0. Otherwise, the log page stored in the rs_locater system table may become invalid. Starting the RepAgent may then cause Adaptive Server to register data corruption and to produce errors such as 605 and 813.

- Transactions that execute after you have turned off the secondary truncation point are not transferred to the Replication Server. Therefore, primary and replicate databases may not be in synch.

For this reason, after you have truncated the log and after the Replication Server has been brought up successfully, you may have to alter replication definitions, drop and re-create subscriptions, and re-materialize the data in the replicate database. New columns will be null until the data is re-materialized.

If a relatively small number of transactions did not transfer to the Replication Server, you may instead choose to use the rs_subcmp program to reconcile the primary and replicate databases.

See also

admin get_generation, dbcc dbrepair, rs_subcmp, rs_zeroltm, sp_config_rep_agent

set replication

Description

A Transact-SQL command that enables or disables replication of data definition language (DDL) and/or data manipulation language (DML) commands to the standby database for the current isql session.

Syntax

set replication ['on' | force_ddl | 'default' | 'off']

Parameters

on
: Enables replication of DML commands for tables marked with sp_setreptable, if sp_reptostandby is set to “none.” If sp_reptostandby is set to “L1” or “all,” enables replication of DML and DDL commands to the standby database. This is the default setting.

force_ddl
: Always enables replication of DDL commands for the current session. If sp_reptostandby is set to “L1” or “all,” DML commands are replicated for all user tables. If sp_reptostandby is set to “none,” DML commands are replicated for tables marked with sp_setreptable.

Note Beginning with Replication Server version 12.0, force_ddl as used in the command set replication force_ddl is no longer a reserved word. This does not affect set replication force_ddl functionality; you no longer have to use double quotes when using force_ddl in other object names.

---

default
: Turns off force_ddl and returns set replication status to “on”—the default.

off
: Turns off replication of marked tables and user stored procedures for the current session. No DML commands and no DDL commands are copied to the standby or replicate database.

Usage

- set replication requires Adaptive Server version 11.5 or later databases.

Permissions

set replication requires “sa” or “dbo” permission and replication_role.

See also

sp_reptostandby, sp_setreptable

sp_configure 'enable rep agent threads'

Description

Enables or disables RepAgent thread integration in the Adaptive Server.

Syntax

```
sp_configure 'enable rep agent threads'[, 1 | 0 ]
```

Parameters

1
: Enables RepAgent integration for the data server.

0
: Disables RepAgent integration for the data server.

Usage

- Use sp_configure 'enable rep agent threads' to enable RepAgent for Adaptive Server version 12.0 or later databases.
- Use sp_configure 'enable rep agent threads' without options to display the current value, default value, and most recently changed value.
- Enable RepAgent in this order:
  - sp_addserver – identifies the Adaptive Server for RepAgent. You need to do this only once.
  - sp_configure 'enable rep agent threads' – enables the data server for RepAgent. You need to do this only once.
  - sp_config_rep_agent – enables the database for RepAgent.

  Refer to the *Adaptive Server Enterprise Reference Manual* for more information about sp_addserver.

Permissions

sp_configure requires “sa” or “sso” permission to modify configuration parameters.

Anyone can execute sp_configure to display information about parameters and their values.

See also

sp_config_rep_agent and more information about sp_configure in the *Adaptive Server Enterprise Reference Manual*.

sp_config_rep_agent

Description

Changes or displays the configuration parameters for the RepAgent thread for an Adaptive Server database.

Syntax

```
sp_config_rep_agent [dbname
          [, {'enable', 'repserver_name',
               'repserver_username'. 'repserver_password'} |
               'disable'[, 'preserve secondary truncpt'] |
               'rs servername', 'repserver_name'
               'rs username', 'repserver_username',
               'rs password', 'repserver_password' |
               'scan batch size', 'no_of_qualifying_log_records' |
               'scan timeout', 'scan_timeout_in_seconds' |
               'retry timeout', 'retry_timeout_in_seconds' |
               'skip ltl errors', {'true' | 'false'} |
               'batch ltl', {'true' | 'false'} |
               'send warm standby xacts', {'true' | 'false'} |
               ‘send buffer size’, {‘2K’ | ‘4K’ |’8K’ | ‘16K’} |
               'connect dataserver','connect_dataserver_name' |
               'connect database', 'connect_database_name' |
               'send maint xacts to replicate',{'true' | 'false'} |
               ‘send structured oqids’, {‘true’ | ‘false’} |
               'short ltl keywords', {'true' | 'false'} |
               'security mechanism', 'mechanism_name' |
               'unified login', {'true' | 'false'}|
               'mutual authentication', {'true' | 'false'} |
               'msg confidentiality', {'true' | 'false'} |
               'msg integrity', {'true' | 'false'} |
               'msg replay detection', {'true' | 'false'} |
               'msg origin check', {'true' | 'false'} |
               'msg out-of-sequence check', {'true' | 'false'}
               'skip unsupported features', {'true' | 'false'} |
               'schema cache growth factor', 'growth_factor' |
              'ha failover', {'true' |'false'} |
               'data limits filter mode', {'off' | 'stop' | 'skip' |'truncate'} |
               'priority', {'4' | '5' | '6'} |
              ‘startup delay’, 'delay'
             ‘net password encryption’, {‘true’ | ‘false’}]
```

Parameters

dbname

The name of the database for which you want to configure RepAgent.

enable

Marks the database as using RepAgent and sets the secondary truncation point to valid.

This command encodes the Replication Server password and inserts the Replication Server name, Replication Server user, and encoded password into the sysattributes table of the specified database.

repserver_name
The name of the Replication Server to which RepAgent connects and transfers log transactions.

repserver_username
The user name that RepAgent thread uses to connect to Replication Server.

repserver_password
The password that RepAgent uses to connect to Replication Server.

If network-based security is enabled and you want to establish unified login, you must specify NULL for *repserver_password* when enabling RepAgent at the database.

rs servername, *repserver_name*
The new or existing name of the Replication Server to which RepAgent connects and transfers log transactions.

rs username, *repserver_username*
The new or existing user name that RepAgent thread uses to connect to Replication Server.

rs password, *repserver_password*
The new or existing password that RepAgent uses to connect to Replication Server.

disable
Unmarks the database as using RepAgent. Use preserve secondary truncpt to retain the secondary truncation point. The default sets the secondary truncation point to IGNORE; that is, it disables it.

Use disable only when downgrading the Replication Server to an earlier release or changing the primary database to another status. This command truncates all RepAgent entries in the sysattributes table.

scan batch size '*no_of_qualifying_records*'
Specifies the maximum number of log records to send to Replication Server in each batch. When the maximum number of records is met, RepAgent asks Replication Server for a new secondary truncation point. The default is 1000 records.

scan timeout '*scan_timeout_in_seconds*'

Specifies the number of seconds that RepAgent sleeps once it has scanned and processed all records in the transaction log and Replication Server has not yet acknowledged previously sent records by sending a new secondary truncation point. RepAgent again queries Replication Server for a secondary truncation point after scan timeout seconds. The default is 15 seconds.

RepAgent continues to query Replication Server until Replication Server acknowledges previously sent records either by sending a new secondary truncation point or extending the transaction log.

If Replication Server has acknowledged all records and no new transaction records have arrived at the log, RepAgent sleeps until the transaction log is extended.

retry timeout '*retry_timeout_in_seconds*'

Specifies the number of seconds RepAgent sleeps before attempting to reconnect to Replication Server after a retryable error or when Replication Server is down. The default is 60 seconds.

skip ltl errors

Specifies whether RepAgent ignores errors in LTL commands. This option is normally used in recovery mode. When set to “true,” RepAgent logs and then skips errors returned by the Replication Server for distribute commands. When set to “false,” RepAgent shuts down when these errors occur. The default is “false.”

batch ltl

Specifies whether RepAgent sends LTL commands to Replication Server in batches or one command at a time. When set to “true,” the commands are sent in batches. The default is “false.”

send warm standby xacts

Specifies whether RepAgent sends information about maintenance users, schema, and system transactions to the warm standby database. This option should be used only with the RepAgent for the currently active database in a warm standby application. The default is “false.”

send buffer size '*2K*', '*4K*', '*8K*', '*16K*'

Controls the size of the send buffer that RepAgent uses to communicate with Replication Server. increasing the size of the send buffer reduces the number of times RepAgent communicates with Replication Server, but increases the amount of memory used.

The default value is 2K.

connect dataserver '*connect_dataserver_name*'
: Specifies the name of the data server RepAgent uses when connecting to Replication Server in recovery mode. This is the data server name RepAgent uses for the connect source command; it is normally the data server for the primary database.

connect database '*connect_database_name*'
: Specifies the name of the temporary database RepAgent uses when connecting to Replication Server in recovery mode. This is the database name RepAgent uses for the connect source command; it is normally the primary database.

send maint xacts to replicate
: Specifies whether RepAgent should send records from the maintenance user to the Replication Server for distribution to subscribing sites. The default is "false."

send structured oqids
: Specifies whether RepAgent sends origin queue IDs (OQIDs) as structured tokens, which saves space in the LTL and thus improves throughput, or as binary strings. The default value is "false."

short ltl keywords
: Specifies whether RepAgent sends an abbreviated form of LTL to Replication Server, requiring less space and reducing the amount of data sent. The default value is "false."

security mechanism '*mechanism_name*'
: Specifies the network-based security mechanism RepAgent uses to connect to Replication Server.

unified login
: When a network-based security system is enabled, specifies whether RepAgent seeks to connect to other servers with a security credential or password. The default is "false."

mutual authentication
: Specifies whether RepAgent should require mutual authentication checks when connecting to Replication Server. The default is "false." This option is not implemented.

msg confidentiality
: Specifies whether to encrypt all messages sent to Replication Server. The default is "false."

msg integrity

Specifies whether all messages exchanged with Replication Server should be checked for tampering. The default is “false.”

msg replay detection

Specifies whether messages received from Replication Server should be checked to make sure they have not been intercepted and replayed. The default is “false.”

msg origin check

Specifies whether to check the source of each message received from Replication Server. The default is “false.”

msg out-of-sequence check

Specifies whether to check the sequence of messages received from Replication Server. The default is “false.”

skip unsupported features

Instructs RepAgent to skip log records for Adaptive Server features unsupported by the Replication Server. This option is normally used if Replication Server is a lower version than Adaptive Server. The default is “false.”

schema cache growth factor '*growth_factor*'

Controls the duration of time table or stored procedure schema can reside in the RepAgent schema cache before expiring. Larger values mean a longer duration and require more memory. Range is 1 to 10. The default is 1.

ha failover

Specifies whether, when Sybase Failover has been installed, RepAgent automatically starts after server failover. The default is “true.”

data limits filter mode {'off' | 'stop' | 'skip' | 'truncate'}

Specifies how RepAgent handles log records containing new, wider columns and parameters, or larger column and parameter counts, before attempting to send them to Replication Server.

- off – RepAgent allows all log records to pass through.
- stop – RepAgent shuts down if it encounters log records containing widedata.
- skip – RepAgent skips log records containing wide data and posts a message to the error log.
- truncate – RepAgent truncates wide data to the maximum the Replication Server can handle.

Warning! Sybase recommends that you do not use the data_limits_filter_mode, off setting with Replication Server version 12.1 or earlier as this may cause RepAgent to skip or truncate wide data, or to stop.

The default value of data limits filter mode depends on the Replication Server version number. For Replication Server versions 12.1 and earlier, the default value is “stop.” For Replication Server versions 12.5 and later, the default value is “off.”

priority {'4' | '5' | '6'}

Sets relative priority values for individual RepAgents.

- 4 – high priority
- 5 – medium priority
- 6 – low priority

The default value of priority is 5.

startup delay, '*delay*'

This delays the automatic start-up of RepAgent by a specified duration to allow Replication Server to be running before RepAgent attempts to connect to Replication Server. By default, RepAgent starts without any delay during automatic start-up. Setting a value in seconds results in a delay in RepAgent start-up by the specified number of seconds. Default: 0 (zero) seconds.

net password encryption, {‘true’ | ‘false’}

Specifies whether connections with a remote server are to be initiated with a client-side password encryption handshake or with the usual unencrypted password handshake sequence. Default: 'true'.

Examples

Example 1 Enables RepAgent for the pubs2 database. RepAgent connects to repsvr1 with repusr1 and password reppwd1:

```
sp_config_rep_agent pubs2, 'enable', 'repsvr1',
  'repusr1', 'reppwd1'
```

Example 2 Displays configuration information for the pubs2 database:

```
sp_config_rep_agent pubs2
```

```
Parameter Name               Default    Config Value    Run Value
----------------------     ---------  ---------------  -------------
priority                       5          5                5
trace flags                    0          0                0
scan timeout                   15         15               15
retry timeout                  60         60               60
rs username                    n/a        rs1_user         rs1_user
batch ltl                      true       true             true
rs servername                  n/a        rs1              rs1
send buffer size               2k         4k               4k
trace log file                 n/a        n/a              n/a
connect database               n/a        n/a              pdb1
connect dataserver             n/a        n/a              pds1
scan batch size                1000       1000             1000
security mechanism             n/a        n/a              n/a
msg integrity                  false      false            false
unified login                  false      false            false
schema cache growth factor     1          1                1
skip ltl errors                false      false            false
msg origin check               false      false            false
short ltl keywords             false      false            false
msg confidentiality            false      false            false
data limits filter mode        stop       stop             stop
msg replay detection           false      false            false
mutual authentication          false      false            false
send structured oqids          false      false            false
send warm standby xacts        false      false            false
msg out-of-sequence check      false      false            false
skip unsupported features      false      false            false
send maint xacts to replicate  false      false            false
net password encryption        true       true             true
startup delay                  0          5                5
```

Example 3 Displays values for a specific parameter.

```
sp_config_rep_agent pubs2, 'scan batch size'
```

```
Parameter Name              Default    Config Value    Run Value
-------------------------------------------------------------------
scan batch size             1000       1000             1000
```

Example 4 Sets scan_timeout to 60 seconds for the pubs2 database:

```
sp_config_rep_agent pubs2, 'scan timeout', '60'
```

Example 5 Configures RepAgent to wait 50 seconds before starting:

```
sp_config_rep_agent pubs2, 'startup delay', '50'
```

Usage

- Use sp_config_rep_agent to configure RepAgent for Adaptive Server databases.
- Enable RepAgent in this way:
  - sp_addserver – identifies the Adaptive Server for RepAgent. You need to do this only once per screen.
  - sp_configure 'enable rep agent thread' – configures the data server for RepAgent. You need to do this only once per screen.
  - sp_config_rep_agent – configures the database for RepAgent.

  Refer to the *Adaptive Server Enterprise Reference Manual* for more information about sp_addserver.
- After you configure the parameters using sp_config_rep_agent, you must restart RepAgent using sp_start_rep_agent for the new parameters to take effect.
- If you execute sp_config_rep_agent without parameters, Adaptive Server displays the default, configured, and runtime values for all databases that are enabled for RepAgent.

  If you only enter *dbname*, Adaptive Server displays the default, configured, and runtime values for the specified database.
- Properties specified by sp_config_rep_agent are stored in the sysattributes table of the database and have an attribute class of RA.
- Use sp_config_rep_agent to set the RepAgent configuration parameters after you have enabled RepAgent at the data server using sp_configure.
- *repserver_user* must have connect source permission.

Configuring network-based security

Note Network-based security for RepAgent is enabled with sp_configure at the Adaptive Server. See the *Adaptive Server Enterprise System Administration Guide* for more information.

- A security mechanism may not support all security properties. Verify the properties of a security mechanism by executing admin security_property at the Replication Server. For more information, see admin security_property on page 67.
- The security mechanism enabled for the RepAgent must be the same as that enabled for the Replication Server. Security settings at the RepAgent and the Replication Server must be compatible.

If RepAgent setting is	Setting at Replication Server can be
"true"	• "required", or • "not required"
"false"	"not required"

- If unified_login is "true," you must specify the rs_password parameter as NULL when RepAgent is enabled at the database.
- If you specify one or more security settings, but do not specify a security mechanism, Adaptive Server initializes the default mechanism, the first entry in the SECURITY section in *$SYBASE/$SYBASE_ASE/config/libtcl.cfg*.

Permissions

sp_configure_rep_agent requires "sa" or "dbo" permission or replication_role.

See also

sp_configure 'enable rep agent threads', sp_help_rep_agent, sp_start_rep_agent, sp_stop_rep_agent

sp_help_rep_agent

Description

Displays static and dynamic information about a RepAgent thread.

Syntax

sp_help_rep_agent [*dbname*[, 'recovery' | 'config' | 'process' | 'scan' | 'security' | 'all']]

Parameters

dbname
The name of the database with the RepAgent for which you want information.

recovery
Displays recovery status information about the RepAgent.

config
Displays configuration information about the RepAgent.

process
Displays information about the RepAgent process.

scan
Displays log-scanning information about the RepAgent.

security
Displays current settings of the network-based security mechanism.

all
Displays all the preceding information for the RepAgent connected to the specified database.

Examples

Example 1 Displays recovery information.

```
sp_help_rep_agent pubs2, 'recovery'
```

```
Rep Agent Recovery Status
dbname  connect     connect  status    rs servername  rs username
         dataserver  database
 ------  ----------  -------- ------    -------------  -----------
 pubs2   sqlserver1   pubs2    scanning  repsvr1        repusr1
```

Example 2 Displays process information.

```
sp_help_rep_agent pubs2, 'process'
```

```
Rep Agent Process Status
dbname  spid   sleep status  retry count   last error
-----   ----   ------------  ----------    ----------
pubs2   40     not sleeping  0             0
```

Example 3 Displays scanning information.

```
                        sp_help_rep_agent pubs2, 'scan'

Replication Agent Scan status
-----------------------------
dbname start marker end marker current marker log recs scanned oldest trans.
----------------------------------------------------------------------------
pubs2  (472675,13) (278622,0) (265736,16)         890       (472675,13)
```

Usage

- Use sp_help_rep_agent with RepAgent-enabled databases.
- If you execute sp_help_rep_agent without parameters, Adaptive Server displays information about all databases for which RepAgent is enabled.
- Table 5-2 describes the output for the sp_help_rep_agent 'recovery' system procedure.

Table 5-2: Column descriptions for sp_help_rep_agent 'recovery' output

Column	Description
dbname	The name of the database containing archived logs whose data is transferred to the Replication Server during recovery.
connect dataserver	The name of the original data server with the database whose transaction logs were transferred to Replication Server in normal mode. This information is included in the LTL connect source command delivered to Replication Server.
connect database	The name of the original database whose transaction logs were transferred to Replication Server in normal mode. This information is included in the LTL connect source command delivered to Replication Server.
status	Indicates RepAgent activity. Status values are: • "not running" – RepAgent is not running. • "not active" – RepAgent is not in recovery mode. • "initial" – RepAgent is initializing in recovery mode. • "end of log" – RepAgent is in recovery mode and has reached the end of the transaction log. • "unknown" – none of the above.
rs servername	The name of the Replication Server to which the RepAgent is transferring information. Use this option to override the *sysattributes* setting.
rs username	The login name RepAgent uses to log in to the Replication Server. Use this option to override the *sysattributes* setting.

- Table 5-3 describes the output for the sp_help_rep_agent 'config' system procedure.

Table 5-3: Column descriptions for sp_help_rep_agent 'config' output

Column	Description
dbname	The name of the database with the data RepAgent is transferring to the Replication Server.
auto start	Contains "true" if the RepAgent starts automatically during server start-up. Otherwise contains "false."
rs servername	The name of the Replication Server to which RepAgent is transferring log transactions.
rs username	The login name the RepAgent thread uses to log in to the Replication Server. The login name must have been granted connect source permission in the Replication Server.
scan batch size	The maximum number of log records sent to Replication Server in each batch. The default is 1000.
scan timeout	The number of seconds that RepAgent sleeps when it has scanned and processed all records in the transaction log and Replication Server has not yet acknowledged previously sent records by sending a secondary truncation point. The default is 15 seconds.
retry timeout	The number of seconds RepAgent sleeps before attempting to reconnect to Replication Server after a retryable error or when Replication Server is down. The default is 60 seconds.
skip ltl errors	Contains "true" if RepAgent ignores errors in LTL commands. Contains "false" if RepAgent shuts down when these errors occur. skip ltl errors is normally set to "true" in recovery mode. The default is "false."
batch ltl	Contains "true" if RepAgent batches LTL commands and sends them to Replication Server. Contains "false" if LTL commands are sent to Replication Server as soon as they are formatted. The default is "false."
send warm standby xacts	Contains "true" if RepAgent submits schema, system xacts, and all updates, including updates made by the maintenance user, to the Replication Server for application to the standby database in a warm standby application. Contains "false" if RepAgent is not submitting updates to the standby database. The default is "false."
connect dataserver	The name of the data server RepAgent connects to Replication Server as when running in recovery mode. If RepAgent is not running in recovery mode, contains the name of the data server of the *dbname* database.
connect database	The name of the database RepAgent connects to Replication Server as when running in recovery mode. If RepAgent is not running in recovery mode, contains the *dbname* database name.

Column	Description
send maint commands to replicate	Contains "true" if RepAgent sends records from the maintenance user to replicate databases. Contains "false" if RepAgent does not send records form the maintenance user to replicate databases. The default is "false."
ha failover	Specifies whether, when Sybase Failover has been installed, RepAgent starts automatically after server failover. The default is "true."
skip unsupported features	Instructs RepAgent to skip log records for Adaptive Server features unsupported by the Replication Server. This option is normally used if Replication Server is an earlier version than Adaptive Server. The default is "false."
short ltl keywords	Specifies whether RepAGent sends an abbreviated form of LTL to Replication Server, requiring less space and reducing the amount of data sent. The default value is "false."
send buffer size	Controls the size of the send buffer that RepAgent uses to communicate with Replication Server. increasing the size of the send buffer reduces the number of times RepAgent communicates with Replication Server, but increases the amount of memory used. Values are "2K," "4K," "8K," and "16K." The default value is "2K."
priority	Sets relative priority values for individual RepAgents. • 4 (high priority) • 5 (medium priority) • 6 (low priority) The default value of priority is 5.
send structured oqids	Specifies whether RepAgent sends origin queue IDs (OQIDs) as structured tokens, which saves space in the LTL and thus improves throughput, or as binary strings. The default value is "false."
data limits filter mode	Specifies how RepAgent handles log records containing new, wider columns and parameters, or larger column and parameter counts, before attempting to send them to Replication Server. • off – RepAgent allows all log records to pass through. • stop – RepAgent shuts down if it encounters log records containing wide data. • skip – RepAgent skips log records containing wide data and posts a message to the error log. The default value of data_limits_filter_mode depends on the Replication Server version number. For Replication Server versions 12.1 and earlier, the default value is "stop." For Replication Server versions 12.5 and later, the default value is "off."

Column	Description
schema cache growth factor	Controls the duration of time table or stored procedure schema can reside in the RepAgent schema cache before expiring. Larger values mean a longer duration and require more memory. Range is 1 to 10. The default is 1.

- Table 5-4 describes the output for the sp_help_rep_agent 'process' system procedure.

Table 5-4: Column descriptions for sp_help_rep_agent 'process' output

Column	Description
dbname	The name of the Replication Server to which RepAgent is transferring log transactions.
sleep status	Sleep status values are: • "waiting for rewrite" – RepAgent is waiting for a two-phase commit transaction to commit. • "end of log" – RepAgent is at the end of the log, waiting for it to be extended. • "connect retry" – RepAgent is waiting before attempting a connection to Replication Server. • "not sleeping" – none of the above. RepAgent is active.
retry count	The number of times RepAgent has unsuccessfully attempted to connect to Replication Server since the last successful connection.
spid	The PID in the dataserver.
last error	The error number of the last Replication Server or connection error.

- Table 5-5 describes the output for the sp_help_rep_agent 'scan' system procedure.

Table 5-5: Column descriptions for sp_help_rep_agent 'scan'output

Column	Description
dbname	The name of the Replication Server to which RepAgent is transferring log transactions.
start marker	Identifies the first record scanned in current batch.
end marker	Identifies the last record to be scanned in current batch.
current marker	Identifies the record currently being scanned.
log recs scanned	The number of log records RepAgent has scanned in the current batch.
oldest transaction	Identifies the oldest transaction in the batch currently being scanned.

- Table 5-6 describes output for the sp_help_rep_agent 'security' stored procedure.

Table 5-6: Column descriptions for sp_help_rep_agent 'security' output

Column	Description
dbname	The name of the Replication Server to which RepAgent is transferring log transactions.
security mechanism	The name of the enabled security mechanism.
unified login	Specifies whether RepAgent seeks to connect to Replication Server with a credential (“true”) or a password (“false”). The default is “false.”
mutual authentication	Specifies whether RepAgent uses mutual authentication checks when connection to Replication Server. The default is “false.”
msg confidentiality	Specifies whether RepAgent uses message encryption on all data sent to Replication Server. The default is “false.”
msg integrity	Specifies whether RepAgent uses message integrity checks on all data exchanged with Replication Server. The default is “false.”
msg replay detection	Specifies whether RepAgent checks to detect whether data has been captured and replayed by an intruder. The default is “false.”
msg origin check	Specifies whether RepAgent verifies the source of data sent from Replication Server. The default is “false.”
msg out-of-sequence	Specifies whether RepAgent verifies that messages received from Replication Server are received in the order sent. The default is “false.”

Permissions

sp_help_rep_agent requires “sa” or “dbo” permission or replication_role.

See also

sp_config_rep_agent, sp_start_rep_agent, sp_stop_rep_agent

sp_reptostandby

Description

Marks or unmarks database for replication to the standby database. Enables replication of supported schema changes and data changes to user tables.

Syntax

sp_reptostandby *dbname* [, 'L1' | 'all' | 'none'] [, use_index]

Parameters

dbname
The name of the active database.

L1
Sets the schema replication feature set support level to the support level first introduced in Adaptive Server version 12.0. If you upgrade the Adaptive Server to a later version that implements a higher support level (that is, L2, L3, and so on) the support level will remain at the Adaptive Server version 12.0 support level. To date, only support level L1 has been implemented in Adaptive Server version 12.0 and later.

all
Sets the schema replication feature set support level to the highest support level implemented by the current Adaptive Server. If you upgrade the Adaptive Server to a later version, the highest support level implemented by the later version is enabled automatically.

none
Unmarks all database tables for replication and turns off data and schema replication to the standby database.

Note If you turn replication off using sp_reptostandby with the none keyword, Adaptive Server locks all user tables in exclusive mode and writes log records for all tables that are unmarked for replication. This can be time-consuming if there are many user tables in the database.

---

use_index
Marks the database to use an index for replication on text, unitext, image, or rawobjects columns.

Examples

Sets the replication status for pubs2 to all and creates a global index on the text and image pointers:

```
sp_reptostandby pubs2,'all','use_index'
```

Usage

- Use sp_reptostandby with Adaptive Server version 11.5 or later databases. You also must enable RepAgent at the active and standby databases.

- Copies data manipulation language (DML) commands, supported data definition language (DDL) commands, and supported system procedures to the standby database.

  The supported DDL commands are:

  - alter table
  - alter key
  - create default
  - create index
  - create key
  - create procedure
  - create rule
  - create table
  - create trigger
  - create view
  - drop default
  - drop index
  - drop procedure
  - drop rule
  - drop table
  - drop trigger
  - drop view
  - grant
  - revoke

  The supported system procedures are:

  - sp_addalias
  - sp_addgroup
  - sp_addmessage
  - sp_adduser
  - sp_addtype

    - sp_bindefault
    - sp_bindmsg
    - sp_bindrule
    - sp_changegroup
    - sp_chgattribute
    - sp_commonkey
    - sp_config_rep_agent
    - sp_dropalias
    - sp_dropgroup
    - sp_dropkey
    - sp_dropmessage
    - sp_droptype
    - sp_dropuser
    - sp_encryption
    - sp_foreignkey
    - sp_primarykey
    - sp_procxmode
    - sp_recompile
    - sp_rename
    - sp_setrepcol
    - sp_setreplicate
    - sp_setreptable
    - sp_unbindefault
    - sp_unbindmsg
    - sp_unbindrule
- If the database is the master database, the DDL commands and system procedures that are supported for replication in a user database are not supported for replication in the master database.

  If the database is the master database, the supported DDL commands are:

    - alter role
    - create role
    - drop role
    - grant role
    - revoke role

    If the database is the master database, the supported system procedures are:

    - sp_addlogin
    - sp_displaylevel
    - sp_droplogin
    - sp_locklogin
    - sp_modifylogin
    - sp_password
    - sp_passwordpolicy
    - sp_role

    If a DDL command or system procedure contains password information, the password information is sent through the replication environment using the ciphertext password value stored in source ASE system tables.

- sp_reptostandby marks the database for replication to the warm standby database. It does not enable replication to replicate databases.
- After sp_reptostandby has been executed and the warm standby enabled, you cannot selectively turn off replication for individual database objects. You can use the set replication command to control replication of DDL and DML commands and procedures for the isql session. See set replication for more information.
- By default, sp_reptostandby marks text, unitext, or image data as replicate_if_changed. You cannot change the status to always_replicate or do_not_replicate.
- If the warm standby application includes normal replication, text, unitext, or image data columns may be treated as always_replicate or replicate_if_changed.

    - If text, unitext, or image columns marked by sp_setreptable are specified always_replicate (the default), all text, unitext, or image columns are treated as always_replicate.
    - If text, unitext, or image columns are specified by sp_setrepcol as do_not_replicate or replicate_if_changed, all text, unitext, or image columns are treated as replicate_if_changed.
- When the database contains one or more large tables holding text, unitext, image, or rawobject columns, the internal process performed by sp_reptostandby may take a long time. To speed up the process, you can use use_index which creates a global nonclustered index for every text, unitext, image, or rawobject column of tables not explicitly marked for replication.
- With use_index, a shared-table lock is held while the nonclustered index is created.
- When you run sp_reptostandby with the none option, and the database is initially marked to use indexes for replication, all those indexes created for replication are dropped.

Restrictions and requirements

- The standby database must be of the same or later release level than the active database. Both databases must have the same disk allocations, segment names, and roles. Refer to the *Adaptive Server Enterprise System Administration Guide* for details.
- Login information is not replicated to the standby database.
- Replication of commands or procedures containing the name of another database will fail if the named database does not exist in the standby server.
- Supported DDL commands, such as create table, may not contain local variables.
- Some commands that are not copied to the standby database:
    - select into and update statistics
    - Database or configuration options such as sp_dboption and sp_configure
- If the database is the master database:
    - User tables and user stored procedures are not replicated.

- - The target database cannot be materialized with dump or load. Use other methodologies, such as bcp, where the data can be manipulated to resolve inconsistencies.
  - Both the source ASE server and target ASE server must support the master database replication feature.
  - Both the source ASE server and the target ASE server must have the same hardware architecture type (32-bit versions and 64-bit versions are compatible) and the same operating system (different versions are compatible).
- If the master database is replicated, the following system procedures must be executed in the master database:
  - sp_addlogin
  - sp_displaylevel
  - sp_droplogin
  - sp_locklogin
  - sp_modifylogin
- You cannot use drop index to manually drop indexes created for text, unitext, image, or rawobject replication. You can use only the supported replication stored procedures sp_reptostandby, sp_setreptable, and sp_setrepcol to change the replication index status.

Permissions

sp_reptostandby requires “sa” or “dbo” permission or replication_role.

See also

set replication, sp_setrepcol, sp_setreptable,sp_setreplicate, sp_setrepproc

sp_setrepcol

Description

Sets or displays the replication status for text, unitext, or image columns.

Syntax

```
sp_setrepcol table_name [, {column_name | null}
        [, {do_not_replicate | always_replicate | replicate_if_changed}]]
       [, use_index]
```

Parameters

table_name
The name of the replicated table. You must enable replication for the table using sp_setreptable before you execute sp_setrepcol.

column_name
The name of a text, unitext, or image column in the table. Specify null for the column name to set the replication status of all text, unitext, or image columns in the table.

do_not_replicate
Prevents Adaptive Server from logging replication information for the text, unitext, or image column. If the column has previously been marked to use an index for replication, setting do_not_replicate removes the index.

always_replicate
Causes Adaptive Server to log replication information for the text, unitext, or image column when any column in the row changes. This status adds overhead for replicating text, unitext, or image columns that do not change; however, it protects against data inconsistency from row migration or changes during non-atomic materialization.

replicate_if_changed
Causes Adaptive Server to log replication information for the text, unitext, or image column only when the text, unitext, or image column data changes. This status reduces overhead, but it may lead to data inconsistency from row migration or changes during non-atomic materialization.

use_index
Marks the column to use an index for replication on text, unitext, image, or rawobjects columns.

Examples

Example 1 Displays the replication status for all text, unitext, or image columns in the au_pix table. au_pix must be marked for replication using sp_setreptable.

```
sp_setrepcol au_pix
```

Example 2 Displays the replication status for the pic column in the au_pix table. pic must be a text, unitext, or image datatype column.

```
sp_setrepcol au_pix, pic
```

Example 3 Specifies that the pic column (image datatype) in the au_pix table should have the replicate_if_changed status. (In this particular table in the pubs2 database, there are no other text, unitext, or image columns.)

```
sp_setrepcol au_pix, pic, replicate_if_changed
```

Example 4 Specifies that all text, unitext, or image columns in the au_pix table should have the replicate_if_changed status.

```
sp_setrepcol au_pix, null, replicate_if_changed
```

Example 5 Marks the column t (text datatype) as replicate_if_changed and uses an index for replication:

```
sp_setrepcol t1, t, replicate_if_changed, use_index
```

Usage

- Use sp_setrepcol to specify how text, unitext, or image columns are replicated after you have enabled replication for the table with sp_setreptable.
- You can also execute sp_setrepcol with a table name to display the replication status of all of the text, unitext, or image columns in the table, or with the table name and a text, unitext, or image column name to display the replication status of the specified column.
- Using the replicate_if_changed option reduces the overhead of replicating text, unitext, or image columns. However, the following restrictions and cautions apply:
  - If you specify the replicate_if_changed status for a column, any replication definition that includes the column must also have the replicate_if_changed status.
  - If you set the replication status of any column to replicate_if_changed, you cannot set autocorrection to "on" for any replication definition that includes the column.
  - If you use non-atomic subscription materialization and you have set the replicate_if_changed replication status for any text, unitext, or image columns, Replication Server displays a message in the error log file. This message warns you that the data may be inconsistent if an application modified the primary table during subscription materialization.
  - If your application allows rows to migrate into a subscription and you have set the replicate_if_changed replication status for any text, unitext, or imagecolumn, Replication Server displays a warning message in the error log when the row migrates into the subscription and the text or image data is missing.

If a text, unitext, or image column with the replicate_if_changed status was not changed in an update operation at the primary table and the update causes the row to migrate into a subscription, the inserted row at the replicate table will be missing the text, unitext, or image data. Run the rs_subcmp program to reconcile the data in the replicate and primary tables.

Row migration can occur when subscriptions have where clauses. Updating a column specified in the subscription where clause can cause a row to become valid for, or migrate into, the subscription.

When this happens, Replication Server must execute an insert in the replicate database. An insert requires values for all of the columns, including text, unitext, or image columns that did not change in the primary database.

- When tables are marked with sp_reptostandby, you cannot change the replication status of text, unitext, or image columns using sp_setrepcol; text, unitext, and image columns are always treated as replicate_if_changed.
- If the warm standby application includes normal replication and you have marked tables with sp_reptostandby and sp_setreptable, text, unitext, or image data columns may be treated as always_replicate or replicate_if_changed.
    - If text, unitext, or image columns marked by sp_setreptable are specified always_replicate (the default), all text, unitext, and image columns are treated as always_replicate.
    - If text, unitext, or image columns are specified by sp_setrepcol as do_not_replicate or replicate_if_changed, all text, unitext, or image columns are treated as replicate_if_changed.
- The order of the precedence on the index status is: column, table, database. If the table is marked to use indexes on text, unitext, image or rawobject columns, but you do not want to use indexes in one of the columns, the column status overrides the table status.
- You cannot use drop index to manually drop indexes created for text, unitext, image, or rawobject replication. You can use only the supported replication stored procedures sp_reptostandby, sp_setreptable, and sp_setrepcol to change the replication index status.

Permissions

sp_setrepcol requires “sa” or “dbo” permission or replication_role.

See also

sp_reptostandby, sp_setreplicate, sp_setreptable

sp_setrepdefmode

Description	Changes or displays the owner status of tables marked for replication.
Syntax	sp_setrepdefmode *table_name* [,{'owner_on' | 'owner_off'}]

Parameters

table_name
The name of a table in the current database that has been marked for replication with sp_setreptable.

owner_on
Changes the owner status of the table so the table name and owner name are considered when the table is marked for replication. Enables replication of multiple tables of the same name with different owners.

owner_off
Changes the owner status of the table so that only the table name is considered when the table is marked for replication.

Usage

- Use sp_setrepdefmode with RepAgent-enabled Adaptive Server databases.
- If sp_setrepdefmode is executed with the table name only, it displays the current mode of the table—"owner on" or "owner off."
- Use sp_setrepdefmode to change the mode of the table. You cannot change the owner mode of tables with sp_setreptable.
- If the owner_off option is supplied and the current mode of the table is "owner on," sp_setrepdefmode checks that the table name is unique among all replicated tables in "owner off" mode. If the name is unique, sp_setrepdefmode changes the table mode to "owner off." If the name is not unique, the procedure fails.

Permissions

sp_setrepdefmode requires "sa" or "dbo" permission or replication_role.

See also

sp_setreptable

sp_setreplicate

Description

This system procedure enables or disables replication for an Adaptive Server table or stored procedure. It also displays the current replication status of a table or stored procedure.

Note This system procedure is still supported, but its capabilities have been incorporated into the system procedures sp_setreptable and sp_setrepproc. sp_setreplicate sets the replication status of columns with text, unitext, or image datatype to do_not_replicate. To replicate text, unitext, or image columns, use the sp_setreptable system procedure instead of sp_setreplicate. To specify individual text, unitext, or image columns for replication, use sp_setrepcol after using sp_setreplicate or sp_setreptable.

Syntax

sp_setreplicate [*object_name* [, {'true' | 'false'}]]

Parameters

object_name
is the name of a table or stored procedure in the current database.

true
enables replication for the table or stored procedure.

false
disables replication for the table or stored procedure.

Examples

Example 1 Displays the replication status for all of the tables and stored procedures in the current database.

```
sp_setreplicate
```

Example 2 Displays the replication status for the publishers table.

```
sp_setreplicate publishers
```

Example 3 Enables replication for the publishers table.

```
sp_setreplicate publishers, 'true'
```

Usage

- Use sp_setrepproc to enable or disable replication of stored procedures when you are using function replication definitions. Use either sp_setrepproc or sp_setreplicate to enable or disable replication of stored procedures when you are using table replication definitions.
- Use sp_setreplicate with no parameters to display a list of replicated tables or stored procedures in the database.
- Use sp_setreplicate *object_name* without true or false to display the current replication status of the table or stored procedure.

- If you use sp_reptostandby to mark a table for implicit replication to the standby database, text, unitext, or image columns set by sp_setreplicate or sp_setrepcol to do_not_replicate are treated as replicate_if_changed. Columns set as always_replicate or replicate_if_changed are treated as marked.
- Because Adaptive Server Enterprise starts a transaction to execute replicated stored procedures, it is important to keep these point in mind when you design procedures:
    - If a replicated stored procedure contains DDL commands (for example, CREATE TABLE), Adaptive Server Enterprise generates an error unless the database option “DDL-in-Tran” is enabled on the database.
    - If the replicated stored procedure contains transactions and rollback commands that roll back the transaction, the rollback command rolls back the execution of the entire procedure.
    - Because of the outer transaction, Adaptive Server Enterprise holds all the locks until the execution of the procedure is complete.

See also sp_setrepcol, sp_setrepproc, sp_setreptable

sp_setrepproc

Description

Enables or disables replication for a stored procedure or displays the current replication status of a stored procedure.

Syntax

```
sp_setrepproc [proc_name [, {'function' | 'table' | 'false'} [,
        {'log_current' | 'log_sproc' }]]]
```

Parameters

proc_name
The name of a stored procedure in the current database.

function
Enables replication for a stored procedure associated with a function replication definition.

table
Enables replication for a stored procedure associated with a table replication definition. This option is equivalent to executing sp_setreplicate on the procedure.

false
Disables replication for the stored procedure.

log_current
Logs the execution of the stored procedure you are replicating in the current database, not the database where the replicated stored procedure resides.

log_sproc
Logs the execution of the stored procedure you are replicating in the database where the stored procedure resides, not in the current database. log_sproc is the default.

Examples

Example 1 Displays the replication status for all of the stored procedures in the current database. For each procedure, indicates whether it is enabled for replication at all, enabled using a function replication definition, or enabled using a table replication definition.

```
sp_setrepproc
```

Example 2 Displays the replication status for the upd_pubs stored procedure. Indicates whether the stored procedure is enabled for replication at all, enabled using a function replication definition, or enabled using a table replication definition.

```
sp_setrepproc upd_pubs
```

Example 3 Enables replication for the upd_pubs stored procedure for use with a function replication definition. The execution of upd_pubs is logged in the database where upd_pubs resides.

```
sp_setrepproc upd_pubs, 'function'
```

Example 4 Enables replication for the upd_pubs stored procedure for use with a table replication definition. The execution of upd_pubs is logged in the database where upd_pubs resides.

```
sp_setrepproc upd_pubs, 'table'
```

Example 5 Enables replication for the upd_pubs stored procedure for use with a table replication definition. The execution of upd_pubs is logged in the current database.

```
sp_setrepproc upd_pubs, 'table', 'log_current'
```

Example 6 Enables replication for the upd_publ stored procedure for use with a table replication definition. The execution of upd_pubs is logged in the database where upd_pubs resides.

```
sp_setrepproc upd_pubs, 'table', 'log_sproc'
```

Usage

- Use sp_setrepproc with no parameters to display all replicated stored procedures in the database.
- Use sp_setrepproc *proc_name* with no other parameters to display the current replication status of the stored procedure.
- If you are using Adaptive Server version 11.5 or later, supported DDL commands and stored procedures executed inside a user stored procedure are copied to the standby database if the procedure is enabled for replication with sp_setrepproc.

  Supported DDL commands and stored procedures executed inside a user stored procedure are not copied to the standby database if the procedure is not enabled for replication with sp_setrepproc.
- Because Adaptive Server starts a transaction to execute replicated stored procedures, keep these points in mind when you design procedures:
  - If a replicated stored procedure contains DDL commands (for example, CREATE TABLE), Adaptive Server Enterprise generates an error unless the database option “DDL-in-Tran” is enabled on the database.
  - If the replicated stored procedure contains transactions and rollback commands that roll back the transaction, the rollback command rolls back the execution of the entire procedure.
  - Because of the outer transaction, Adaptive Server holds all the locks until the execution of the procedure is complete.

See also sp_reptostandby, sp_setreplicate, sp_setreptable

sp_setreptable

Description

Enables or disables replication for an Adaptive Server table or displays the current replication status of a table.

Syntax

sp_setreptable [*table_name* [, {'true' | 'false'} [, {owner_on | owner_off | null}] [, use_index]]]

Parameters

table_name
The name of the table marked for replication.

true
Enables replication for the table.

false
Disables replication for the table.

owner_on
Sets the mode of the table so that both the table name and owner name are considered when the table is marked for replication. Enables tables with the same name but different owner be replicated. This option is for Adaptive Server version 11.5 and later databases.

owner_off
Sets the mode of the table so that only the table name is considered when the table is marked for replication. This is the default. It ensures that the name for each table marked for replication is unique. This option is for Adaptive Server version 11.5 and later databases.

null
Sets the default value of owner_off when you pass it to the owner parameter.

use_index
Marks the table to use an index for replication on text, unitext, image, or rawobjects columns.

Examples

Example 1 Displays the replication status for all of the tables in the current database:

```
sp_setreptable
```

Example 2 Displays the replication status for the publishers table:

```
sp_setreptable publishers
```

Example 3 Enables replication for the publishers table:

```
sp_setreptable publishers, 'true'
```

Example 4 Allows multiple tables named publishers each owned by different users to be replicated:

```
sp_setreptable publishers, 'true', owner_on
```

Example 5 Replicates table named publishers belonging to owner dbo and stored in database pubs2:

```
sp_setreptable 'pubs2.dbo.publishers', 'true', owner_on
```

Example 6 Marks the table for replication to use indexes on the text, unitext, image, and rawobject columns, and sets owner status to “off”:

```
sp_setreptable t1, true, null, use_index
```

Example 7 Removes the replication status of table t1, and drops the replication indexes if t1 was initially marked for replication to use indexes:

```
sp_setreptable t1, 'false'
```

Usage

- Use sp_setreptable with no parameters to display a list of replicated tables in the database.
- Use sp_setreptable *table_name* without true or false to display the current replication status of the table.
- When you include the owner_on option, multiple tables with the same table name but different owners may be replicated to replicate and warm standby databases. Make sure that the replication definition on the table also includes owner information or replication may fail.
- If a table has been marked for replication with sp_setreptable, you can change the owner mode with the sp_setrepdefmode system procedure.
- The replication index status order of precedence is: column, table, database. For example, in a database marked for replication using indexes, the table’s status overrides the index status.
- When a large table containing one or more text, unitext, image, or rawobject columns is marked for replication, the internal process is performed in a single transaction and may take a long time. To speed up the process, use the use_index option to create a global nonclustered index for every text, unitext, image, or rawobject column.
- With use_index, a shared-table lock is held while the global nonclustered index is created.
- You cannot use drop index to manually drop indexes created for text, unitext, image, or rawobject replication. You can use only the supported replication stored procedures sp_reptostandby, sp_setreptable, and sp_setrepcol to change the replication index status.

Permissions

sp_setreptable requires “sa” or “dbo” permission or replication_role.

See also	sp_reptostandby, sp_setrepcol, sp_setrepdefmode, sp_setreplicate, sp_setrepproc

sp_start_rep_agent

Description

Starts a RepAgent thread for the specified database.

Syntax

sp_start_rep_agent *dbname*[, {'recovery' | 'recovery_foreground'}
        [, '*connect_dataserver*',
        '*connect_database*'[, '*repserver_name*', *repserver_username*',
        '*repserver_password*']]]

Parameters

dbname
The name of the database for which you want to start a RepAgent.

recovery
Starts the RepAgent in recovery mode, which is used to initiate recovery actions. Recovery mode is used to rebuild queues when queues are lost.

You can also specify the Replication Server name, user name, and password in recovery mode. Specify these parameters to override sysattributes settings.

recovery_foreground
recovery_foreground has the same function as recovery. However, it displays the recovery progress information on screen instead of in the Adaptive Server error log. The recovery is complete once the recovery progress information display ends and the command prompt displays.

connect_dataserver
The name of the data server used to recover offline logs.

connect_database
The name of the database used to recover offline logs.

repserver_name
The name of the Replication Server to which RepAgent connects.

repserver_user_name
The user name that RepAgent uses to connect to Replication Server.

repserver_password
The password that RepAgent uses to connect to Replication Server.

Examples

Example 1 Starts an integrated RepAgent for the pubs2 database. RepAgent connects to the Replication Server specified in sp_config_rep_agent. It starts scanning the transaction log and sends formatted LTL commands to Replication Server.

```
sp_start_rep_agent pubs2
```

Example 2 Starts RepAgent in recovery mode for the pdb2 database connected to the svr2 data server.

```
sp_start_rep_agent pubs2 for_recovery, svr2, pdb2
```

Example 3 Configures RepAgent to print the recovery of database db2 to the client:

```
sp_start_rep_agent db2, recovery_foreground, ds, db1
```

```
RepAgent(5). Starting recovery, processing log records
   between (1018, 0) and (2355, 2).
RepAgent(5). Processed 1000 log records.
RepAgent(5). Processed 2000 log records.
RepAgent(5). Processed 3000 log records.
RepAgent(5). Processed 4000 log records.
RepAgent(5). Processed 5000 log records.
RepAgent(5). Processed 6000 log records.
RepAgent(5). Processed 7000 log records.
RepAgent(5). Processed 8000 log records.
RepAgent(5). Processed 9000 log records.
RepAgent(5). Processed 10000 log records.
RepAgent(5). Processed 11000 log records.
RepAgent(5). Processed 12000 log records.
RepAgent(5). Processed 13000 log records.
RepAgent(5). Processed 14000 log records.
RepAgent(5). Processed 15000 log records.
RepAgent(5). Processed 16000 log records.
RepAgent(5). Processed 17000 log records.
RepAgent(5). Processed 18000 log records.
RepAgent(5). Processed 19000 log records.
RepAgent(5). Processed 20000 log records.
RepAgent(5). Processed 20084 log records, recovery
   complete.
Replication Agent thread is started for database 'db2'.
(return status = 0)
```

Usage

- Use sp_start_rep_agent with RepAgent-enabled databases.
- Use the sp_start_rep_agent command to start up RepAgent after you have enabled it with sp_config_rep_agent. Once you have started RepAgent with sp_start_rep_agent, it will automatically start up after the dataserver is recovered during server startup.
- Autostart is disabled after you have used sp_stop_rep_agent to shut down RepAgent. Reenable it using sp_start_rep_agent.

- For offline recovery, archived transaction logs may be dumped to a temporary recovery database. You can then transfer records in the transaction log of the temporary recovery database to the replicate database. Execute sp_start_rep_agent with either recovery or recovery_foreground, using the temporary data server and database names, to scan the temporary transaction log.

  In recovery, when the RepAgent has completed scanning the transaction log, RepAgent shuts down. After the next transaction dump has been loaded, restart the RepAgent by executing sp_start_rep_agent with the options specified earlier.

Permissions

sp_start_rep_agent requires “sa” or “dbo” permission or replication_role.

See also

sp_help_rep_agent, sp_help_rep_agent, sp_stop_rep_agent

sp_stop_rep_agent

Description

Shuts down the RepAgent thread for the specified database.

Syntax

sp_stop_rep_agent *dbname*[, 'nowait']

Parameters

dbname
The name of the database for which you want to shut down the RepAgent.

nowait
Shuts down the RepAgent immediately, without waiting for executing operations to complete.

The default shuts down RepAgent gracefully at the end of the current batch.

Examples

Shuts down an integrated RepAgent for the pubs2 database. The default shutdown option allows RepAgent to finish processing the current batch.

```
sp_stop_rep_agent pubs2
```

Usage

- Use sp_stop_rep_agent with RepAgent-enabled databases.
- Once you have used sp_stop_rep_agent to shut down RepAgent, it does not automatically start up when the database comes online during server startup. To re-enable automatic startup, execute the sp_start_rep_agent procedure.
- sp_stop_rep_agent is asynchronous and may take some time to execute. Use sp_who to check the status of the RepAgent.

Permissions

sp_start_rep_agent requires “sa” or “dbo” permission or replication_role.

See also

sp_config_rep_agent, sp_help_rep_agent, sp_start_rep_agent

CHAPTER 6 Adaptive Server Stored Procedures

This chapter contains reference pages for the Adaptive Server stored procedures used with Replication Server. Use the RSSD database to execute these stored procedures.

rs_capacity

Description	Helps you estimate stable queue size requirements. Use with the rs_fillcaptable stored procedure.
Syntax	rs_capacity *TranDuration*, *FailDuration*, *SaveInterval*, *MatRows*

Parameters

TranDuration
The duration, in seconds, of the longest transaction. The default is up to 5 seconds.

FailDuration
The length in time, in minutes, that the queue must retain information during a failure. The default is 60 minutes.

SaveInterval
The length of time, in minutes, that messages should be retained after they have been confirmed as received. The default is 1 minute.

MatRows
The number of rows to be materialized in a subscription. The default is 1000 rows.

Examples

For the example scenario described for the rs_fillcaptable stored procedure, use rs_capacity with the following parameters.

```
rs_capacity
 60,    /* TranDuration maximum 60 seconds */
 360,   /* FailDuration 6 hours */
 10,    /* SaveInterval 10 minutes */
 3500   /* Materialize 3500 rows */
```

rs_capacity returns an estimate of the queue sizes needed for each queue. It also gives an estimate of the subscription materialization queue size needed, based on the replication definition and the number of rows to materialize.

Usage

- rs_capacity uses the data in the rs_captable table (created using the rs_fillcaptable stored procedure) to calculate estimates of stable queue size requirements. Execute rs_capacity after you have described replication definitions using rs_fillcaptable.

See also

rs_fillcaptable

rs_delexception

Description	Deletes a transaction in the exceptions log.
Syntax	rs_delexception [*transaction_id*]
Parameters	*transaction_id* The number of the transaction you want to delete.
Examples	Deletes transaction number 1234 from the exceptions log.

```
rs_delexception 1234
```

Usage

- If you do not specify any parameters, rs_delexception displays a summary of transactions in the exceptions log.
- If you supply a valid *transaction_id*, rs_delexception deletes a transaction. You can find the *transaction_id* for a transaction by using either rs_helpexception or rs_delexception with no parameters.

See also rs_helpexception

rs_dump_stats

Description

Extracts Replication Server statistics collected in the RSSD by admin stats to a comma-delimited format.

Syntax

rs_dump_stats ['*comment*']

Parameters

comment

Is an optional description of the statistics being displayed. It appears on the first line of the output file.

Examples

Extracts Replication Server statistics with comment "Stats from 01/31/2006."

```
rs_dump_stats 'Stats from 01/31/2006'
```

The columns of counter data are, in order:

- The timestamp of the observation period
- The number of observations made of the counter during the observation period
- The total of observed values
- The last observed value
- The maximum observed value

Depending on the counter category (see the chapter "Using Counters to Monitor Performance" in the *Replication Server Administration Guide Volume 2* for a description of counter categories), there may be close correlation between the number of observations and total observations, and between the last and maximum observed values. For example, an observer counter simply counts the number of observations of an event—such as the number of times a message is read from a queue. For an observer counter, the number of observations and the total of observed values are the same. Similarly, the last and maximum observed values are both 1 (unless no messages were read in the observation period, in which case both values would be 0).

Note Comments to the right of the output are included to explain the example. They are not part of the rs_dump_stats output.

```
Comment: Stats from 01/31/2006== Provided label
Oct 17 2005  3:13:47:716PM    == End of the first observation period
Oct 17 2005  3:14:24:730PM    == End of the last observation period
2                             == Number of observation periods
0                             == Number of minutes in each obs period.
0 if less than one.(Calculated as the number of minutes between the first
and last obs period, divided by the number of observations.)
16384                        == Number of bytes in an SQM Block to
                             aid calculations
64                           == Number of blocks in an SQM Segment
                             to aid calculations
CM                           == Module Name. See rs_help_counter
                             for a complete list.
13                           == Instance ID. See admin stats for an
                             explanation.
-1                           == Inst Val/Mod Type. Further instance
                             qualification when needed.
dCM                          == Instance description.
CM: Outbound database connection
requests                     == Counter description.
CMOBDBReq                    == Counter display name.
13003           , , 13,  -1  == Counter ID and instance qualifying
                             information.
Oct 17 2005  3:13:47:716PM,  52,
52,  1,  1                   == Counter data. One row output for
                             each observation period. See below for
                             explanation.
Oct 17 2005  3:14:24:730PM,  42,
42,  1,  1
ENDOFDATA                    == End of output for the previous
                             counter
CM: Outbound non-database
connection requests          == Start of output for the next counter
CMOBNonDBReq
13004           , , 13,  -1
Oct 17 2005  3:13:47:716PM,  2,  2,  1,  1
Oct 17 2005  3:14:24:730PM,  2,  2,  1,  1
ENDOFDATA
.
.
.
CM: Time spent closing an ob fadeout conn
CMOBConnFadeOutClose
13019           , , 13,  -1
Oct 17 2005  3:13:47:716PM,  0,  0,  0,  0
Oct 17 2005  3:14:24:730PM,  2,  6,  2,  4
ENDOFDATA
DIST                         == Start of output for the next
                             module/instance
102
-1
DIST, 102 pds03.tpcc
```

```
DIST: Commands read from inbound queue
CmdsRead
30000           , , 102,  -1
Oct 17 2005  3:13:47:716PM,  1,  1,  1,  1
Oct 17 2005  3:14:24:730PM,  1,  1,  1,  1
ENDOFDATA
.
.
.
DSIEXEC: Number of 'message' results
DSIEResMsg
57127           , , 103,  7
Oct 17 2005  3:13:47:716PM,  1,  1,  1,  1
Oct 17 2005  3:14:24:730PM,  1,  1,  1,  1
ENDOFDATA
(return status = 0)            == End of output
```

Usage

- You can capture the output of rs_dump_stats in a text file that can then be analyzed in a spread sheet or other analysis tool.
- If the text file containing the output of rs_dump_stats is too large to load in to the analysis tool, you can split the file into multiple files.
  - Each new file must contain the first seven rows and the last row of the original file.
  - Between the first seven rows and the last row of each new file, insert all rows associated with a given module instance.

  Depending on the analysis tool, it is usually unnecessary to include all instances of one module in the same file.
- rs_dump_stats does not remove or alter statistics saved in the RSSD.
- rs_dump_stats lists counters with no observations, but does not display counter data rows for them. rs_dump_stats displays counter data rows for all counters with at least one observation during the sample period.

See also

rs_helpcounter, admin stats

rs_fillcaptable

Description	Records estimated transaction rates in the rs_captable table for an existing replication definition.
Syntax	rs_fillcaptable *RepDefName*, *InChRateI*, *InChRateD*, *InChRateU*, *OutChRateI*, *OutChRateD*, *OutChRateU*, *InTranRate*, *OutTranRate*, *DelFlag*

Parameters

RepDefName
The name of the replication definition.

InChRateI
The number of inserts per second, including inserts that are not replicated. The default is 15 inserts per second.

InChRateD
The number of deletes per second, including deletes that are not replicated. The default is 15 deletes per second.

InChRateU
The number of updates per second, including updates that are not replicated. The default is 15 updates per second.

OutChRateI
The number of inserts per second, excluding inserts that are not replicated. The default is 15 inserts per second.

OutChRateD
The number of deletes per second, excluding deletes that are not replicated. The default is 15 deletes per second.

OutChRateU
The number of updates per second, excluding updates that are not replicated. The default is 15 updates per second.

InTranRate
The number of transactions per second for the database. The default is 5 transactions per second.

OutTranRate
The number of replicated transactions per second for the database. The default is 5 transactions per second.

DelFlag
Set to “n” or “N” to update the row for the replication definition. Set to “y” or “Y” to delete the row for the replication definition from rs_captable. Set *DelFlag* to “Y” and *RepDefName* to “ALL,” to clear the entire rs_captable table.

Examples

Example 1 In this example scenario, the overall transaction rate in a primary database is 10 transactions per second. Of these 10 transactions, 8 are replicated. The *InTranRate* for the database is 10 and the *OutTranRate* is 8.

There are two replicated transactions, T1 and T2. T1 executes 5 times per second, performs 2 updates to table1, and performs 1 update to table2. T2 executes 3 times per second, performs 2 inserts to table1, and performs 1 insert to table2.

There are two subscriptions in replicate databases, each receiving one half of the replicated data. The transactions are distributed equally across the two subscriptions. Therefore, the outbound estimates are 50 percent of the inbound estimates.

This table summarizes the information from this example scenario:

		table1			***table2***		
		ins	**upd**	**del**	**ins**	**upd**	**del**
Inbound	T1 (5 per second)		10		5		
	T2 (3 per second)	6			3		
	Totals	6	10		8		
Outbound	50% replicated	3	5		4		

To get an estimate of stable queue size requirements for this example scenario, first clear the rs_captable table. Then use rs_fillcaptable with the parameters described above. When you are done, use the rs_capacity stored procedure with the new contents of the rs_captable table.

Example 2 This example clears the rs_captable table.

```
rs_fillcaptable @RepDefName = 'ALL', @DelFlag = 'Y'
```

Example 3 This example fills the rs_captable table with the appropriate values for the first replication definition.

```
rs_fillcaptable
 repdef1, /* replication definition for table1 */
 6,       /* InChRateI */
 0,       /* InChRateD */
 10,      /* InChRateU */
 3,       /* OutChRateI */
 0,       /* OutChRateD */
 5,       /* OutChRateU */
 10,      /* InTranRate */
 8,       /* OutTranRate */
 n        /* DelFlag */
```

Example 4 This example fills the rs_captable table with the appropriate values for the second replication definition.

```
rs_fillcaptable
 repdef2, /* replication definition for table2 */
 8,       /* InChRateI */
 0,       /* InChRateD */
 0,       /* InChRateU */
 4,       /* OutChRateI */
 0,       /* OutChRateD */
 0,       /* OutChRateU */
 10,      /* InTranRate */
 8,       /* OutTranRate */
 n        /* DelFlag */
```

See rs_capacity for more information on using the output derived from these examples to complete the estimate of stable queue size requirements.

Usage

- Use rs_fillcaptable to describe the transactions for each replication definition you want to include in your stable queue estimate.
- rs_fillcaptable maintains a work table named rs_captable that contains estimates of change rates for each replication definition in a database.
- Use the output of rs_fillcaptable as input for the rs_capacity stored procedure.

See also

rs_capacity

rs_helpclass

Description

Displays error classes and function-string classes and their primary Replication Server and, in the case of inherited classes, the parent class.

Syntax

rs_helpclass [*class_name*]

Parameters

class_name

A string of characters that corresponds to an error class or function-string class name. The string must match an entire name or the first part of a name.

Examples

Example 1 Displays information about all error classes and function-string classes for the Replication Server.

```
rs_helpclass
```

```
Function String Class(es)     PRS for CLASS      Parent Class
----------------------------- ------------------ -----------------------
rs_default_function_class     Not Yet Defined.   Base class
rs_sqlserver_function_class   Not Yet Defined.   Base class
sqlserver2_function_class     TOKYO_RS           rs_default_function_class

Error Class(es)                PRS for CLASS
------------------------------ ------------------------------------------
rs_sqlserver_error_class       Not Yet Defined.
```

Example 2 Displays information about the sqlserver2_function_class function-string class.

```
rs_helpclass sqlserver2_function_class
```

Usage

Note Use the command admin show_function_classes to get more information about error classes and function-string classes.

- If you do not enter any parameters, rs_helpclass lists all defined error classes and function-string classes.
- If you supply a *class_name* string, rs_helpclass lists error classes and function-string classes that match *class_name*.
- If a class is not defined at a Replication Server, which is true of default classes for Adaptive Server, rs_helpclass lists it as undefined and tells you how to define it.

rs_helpclassfstring

Description

Displays the function-string information for function strings with function-string-class scope.

Syntax

rs_helpclassfstring *class_name*
[, *function_name*]

Parameters

class_name
The function-string class for which you want to view function strings.

function_name
A string of characters that corresponds to a function name. The string must match an entire function name or the first part of a name.

Examples

Example 1 Displays parameters and function-string text for all functions of the function-string class rs_sqlserver_function_class.

```
rs_helpclassfstring rs_sqlserver_function_class
```

Example 2 Displays the function-string text for the rs_usedb function of rs_sqlserver_function_class.

```
rs_helpclassfstring rs_sqlserver_function_class,
rs_usedb
```

```
Function Name  FString Name FSClass Name
-------------- ------------ ---------------------------
rs_usedb       rs_usedb     rs_sqlserver_function_class

FString Text
--------------------------------------------------------
    use ?rs_destination_db!sys_raw?
```

Usage

- If you do not supply a *function_name* parameter, rs_helpclassfstring displays all function strings defined for all functions of the function-string class.
- If you supply a *function_name* string, rs_helpclassfstring displays function strings that match *function_name*, such as rs_insert, rs_delete, rs_update, and rs_select, or a user-defined function.
- Non-customized, inherited function strings are not displayed for derived function-string classes.

rs_helpcounter

Description

Displays information about counters.

Syntax

```
rs_helpcounter [ { sysmon | duration | observer | monitor
          | must_sample | no_reset | keep_old }
          | module_name [, { short | long } ] | keyword [, { short | long } ] ]
```

Parameters

sysmon
: Specifies those counters most useful for assessing performance and for gathering replication system profile information.

duration
: Specifies all counters that measure duration with time intervals measured in one-hundedths of a second.

observer
: Specifies counters that record the number of times an event occurs. For example, the number of times a message is read from a queue.

monitor
: Specifies counters that record a current value. For example, the size in bytes of the message most recently read from the queue.

must_sample
: Specifies counters that must keep sampling even if sampling is not turned on.

no_reset
: Specifies counters whose values are not reset when admin stats, reset is executed.

keep_old
: Specifies counters that keep both current and previous values.

module_name
: The name of a module: dsi, dsiexec, sqt, cm, dist, rsi, sqm, repagent, and so on.

short
: Tells Replication Server to print the display names, module names, and counter descriptions of counters specified.

long
: Tells Replication Server to print values for every column in the rs_statcounters table.

keyword
Search keyword. Search in the counter long names, the counter display names, and counter descriptions.

Examples

Example 1 Lists all module names, and syntax for using rs_helpcounter.

```
1> rs_helpcounter
2> go

ModuleName
------------------------------
CM
DIST
DSI
DSIEXEC
REPAGENT
RSH
RSI
RSIUSER
SERV
SQM
SQMR
SQT
STS
SYNC
SYNCELE
(12 rows affected)
```

```
How to Use rs_helpcounter
----------------------------------------------------------------
rs_helpcounter  -> Shows module names and help.
rs_helpcounter [ sysmon | duration | observe | monitor
               | must_sample | no_reset | keep_old ]
rs_helpcounter ModuleName    [, {short | long }]
rs_helpcounter keyword       [, { short | long }]
   where "keyword" is part of the counter name, display name or description
   (return status = 0)
```

Example 2 Lists the display names, module names, and counter descriptions for the SQM Reader.

```
rs_helpcounter sqmr, short
```

```
Display Name        Module Name    Counter Description
--------------      ------------    ----------------------------------------
--
BlocksRead          SQMR           Number of 16K blocks read from a stable
                                   queue by an SQM Reader thread.
```

```
ClocksReadCached   SQMR          Number of 16K blocks from cache read by
                                 an SQM Reader thread.
CmdsRead           SQMR          Commands read from a stable queue by an
                                 SQM Reader thread.
SQMRReadTime       SQMR          The amount of time taken for SQMR to read
                                 a block.
SleepsStartQR      SQMR          srv_sleep() calls by an SQM Reader
                                 client due to waiting for SQM thread to
                                 start.
SleepsWriteQ       SQMR          srv_sleep() calls by an SQM read client
                                 due to waiting for the SQM thread to
                                 write.
XNLInterrupted     SQMR          Number of interruptions so far when
                                 reading large messages with partial
                                 read. Such interruptions happen due to
                                 time out, unexpected wakeup, or nonblock
                                 read request, which is marked as
                                 READ_POSTED.
XMLPartials        SQMR          Partial large messages read so far.
XNLReads           SQMR          Large messages read successfully so
                                 far. This does not count partial
                                 messages, or timeout interruptions.
(return status = 0)
```

Usage

- rs_helpcounter lets you search the rs_statcounters system table.
- When used with no parameters, rs_helpcounter prints out a list of modules and syntax.
- For information about counter status and other counter information stored in the RSSD, see the rs_statcounters system table described on page 568.

Permissions

Any user may execute this command.

rs_helpdb

Description
: Provides information about databases that Replication Server knows about.

Syntax
: rs_helpdb [*data_server*, *database*]

Parameters

data_server
The data server with the database whose information you want to display.

database
The database whose information you want to display.

Examples

```
rs_helpdb

dsname                       dbname                    dbid
 --------------------------- ------------------------- ----
 TOKYO_DS                    TOKYO_RSSD                101
 SYDNEY_DS                   SYDNEY_RSSD               102
 TOKYO_DS                    pubs2                     105

controlling_prs                 errorclass
 ------------------------------ -------------------------
 TOKYO_RS                       rs_sqlserver_error_class
 SYDNEY_RS                      rs_sqlserver_error_class
 TOKYO_RS                       rs_sqlserver_error_class

funcclass
 ------------------------------
 rs_sqlserver_function_class
 rs_sqlserver_function_class
 rs_sqlserver_function_class

status
 -------------------------------------------------------------------
 Log Transfer is ON, Distribution is ON
 Log Transfer is ON, Distribution is ON
 Log Transfer is ON, Distribution is ON
```

Usage

- If you do not provide the *data_server* and *database* parameters, rs_helpdb returns results for all of the databases in the rs_databases system table.
- rs_helpdb is executed in a Replication Server's RSSD.
- For each database, rs_helpdb provides the following information:

  dsname – the name of the data server with the database.

  dbname – the name of the database.

dbid – the ID number assigned to uniquely identify the database throughout the replication system.

controlling_prs – the Replication Server that manages the database.

errorclass – the error class Replication Server uses to handle errors returned from the data server for this database.

funcclass – the function-string class used for the database.

status – tells whether log transfer and distribution are on or off for the database.

ltype – the type of database connection (logical or physical).

ptype – the type of database (active database, standby database, or logical connection).

rs_helpdbrep

Description

Displays information about database replication definitions associated with the current Replication Server.

Syntax

rs_helpdbrep [*db_repdef*[, *data_server*[, *database*]]]

Parameters

db_repdef
Specifies the name of the database replication definition.

data_server
Specifies the name of the data server whose database replication definition you want to display.

database
Specifies the name of the database whose database replication definition you want to display.

Examples

Example 1 In this example, Adaptive Server displays the information of all the database replication definitions found in the current Replication Server:

```
rs_helpdbrep
```

```
DB Rep.Def.Name Primary DS.DB Primary RS  Rep.DDL  Rep.Sys. Rep.Tab Rep.Func.
--------------- ------------  ---------   -------- -------  ------- --------
db_rep1          PDS.pdb1      PRS         Yes      Out-List  All    All
db_rep2          PDS.pdb2      PRS         Yes      Out-List  All    All

Rep.Tran. Creation Date
```

```
--------- -----------------
All       May 1 2003 10:58AM
All       May 17 2003 11:16AM
```

Example 2 In this example, Adaptive Server displays information about a single database replication definition, db_rep1.

```
rs_helpdbrep db_rep1
```

```
DB Rep.Def.Name Primary DS.DB Primary RS  Rep.DDL  Rep.Sys. Rep.Tab Rep.Func.
--------------- ------------  ---------   ------- -------  ------- --------
db_rep1          PDS.pdb1      PRS          Yes     Out-List  All     All

Rep.Tran. Creation Date
--------- -----------------
All       May 1 2003 10:58AM

Rep.Type      Owner     Name
------------  --------  -------------
Not Rep.Sys.  .         sp_setrepproc

DBRep.Def.Name DBSub.Name ReplicationDS.DB ReplicateRS Creation Date
-------------- ---------- ---------------- ----------- -----------------
db_rep1        db_sub1    RDS1.rdb1        RRS1        May 1 2003 10:58AM
db_rep1        db_sub2    RDS2.rdb2        RRS2        May 1 2003 10:59AM
```

Usage

- Adaptive Server only displays detail information about named database replication definitions.
- The parameters can contain the wild card '%'. This wild card represents any string. For example, if a string 'abc%' is assigned to *db_repdef*, rs_helpdbrep will list all database replication definition that have a database replication definition name prefixed with 'abc'.

See also rs_helpdbsub

rs_helpdbsub

Description Displays information about database subscriptions associated with the replicate data server.

Syntax rs_helpdbsub [*db_sub*[, *data_server*[, *database*]]]

Parameters

db_sub
Specifies the database subscription.

data_server
Specifies the data server name whose database subscription you want to display.

database
Specifies the database name whose database subscription you want to display.

Examples

In this example, Adaptive Server displays information about a single database subscription, db_sub1:

```
rs_helpdbsub db_sub1, RDS1, rdb1
```

```
DBSub.Name  ReplicateDS.DB  ReplicateRS  Status at RRS  DBRep.Def.Name
----------  --------------  -----------  -------------  ---------------
db_sub1     RDS1.rdb1       RRS1         Validate       db_rep

PrimaryDS.DB  PrimaryRS  Method       Trunc.Table     Creation Date
------------  ---------  -----------  -----------     -----------------
PDS.pdb1       PRS        Bulk Create  Yes              May 2 2003 3:38PM
```

Usage

- If you do not specify any parameters, rs_helpdbsub lists database subscriptions defined in the Replication Server.
- If you supply the *db_sub* parameter only, rs_helpdbsub lists all the database subscriptions defined in the Replication Server that have a database subscription name matching *db_sub*.
- The parameters can contain the wild card '%'. This wild card represents any string. For example, if a string 'abc%' is assigned to *db_sub*, rs_helpdbsub will list all database subscriptions that have a database subscription name prefixed with 'abc'.

See also

rs_helpdbrep

rs_helperror

Description

Displays the Replication Server error actions mapped to a given data server error number.

Syntax

rs_helperror *server_error_number* [, v]

Parameters

server_error_number
A data server error number.

v
Displays the Adaptive Server error message text, if it is available.

Examples

```
rs_helperror 2601, v
```

```
DS Error Num Error Action              Error Class
------------ ------------------------- -----------------------------
2601 Stop Replication                  rs_sqlserver_error_class

Adaptive Server Error Message
----------------------------------------------------------------------
Attempt to insert duplicate key row in object '%.*s' with unique index
 '%.*s'%S_EED
```

Usage

- Error action mappings are displayed for all error classes.
- Use the assign action command to map error actions to data server error numbers.

See also

assign action

rs_helpexception

Description	Displays transactions in the exceptions log.
Syntax	rs_helpexception [*transaction_id*, [, v]]

Parameters

transaction_id
The number of the transaction for which you want help.

v
Includes the text of the transaction in a detailed listing.

Examples

Example 1 Displays summary information on all transactions in the exceptions log.

```
rs_helpexception
```

Example 2 Displays detailed information on transaction number 1234, including the text of the transaction.

```
rs_helpexception 1234, v
```

Usage

- If you do not enter any parameters, rs_helpexception displays a summary list of the transactions in the exceptions log, including all transaction numbers.
- If you supply a valid *transaction_id*, rs_helpexception displays a detailed description of a transaction.
- Use rs_delexception to delete transactions in the exceptions log.

See also

rs_delexception

rs_helpfstring

Description

Displays the parameters and function string text for functions associated with a replication definition.

Syntax

rs_helpfstring *replication_definition*
[, *function_name*]

Parameters

replication_definition
The table or function replication definition for which you want to view functions.

function_name
A string of characters that corresponds to a function name. The string must match an entire function name or the first part of a name.

Examples

Example 1 Displays parameters and function string text for all functions of the replication definition authors_rep.

```
rs_helpfstring authors_rep
```

Example 2 Displays parameters and function string text for the rs_insert function of the replication definition authors_rep.

```
rs_helpfstring authors_rep, rs_insert
```

```
Function String information for Replication Definition.
                       'authors_rep'
Valid Parameters are:
 Parameter Name                 Datatype
 ------------------------------ ------------------------------
 @au_id                         varchar
 @au_lname                      varchar
 @au_fname                      varchar
 @phone                         char
 @address                       varchar
 @city                          varchar
 @state                         char
 @country                       varchar
 @postalcode                    char

Rep.Def.Name  Function Name   FString Name  FSClass Name
-------------- -------------- ------------ ---------------------------
 authors_rep    rs_insert      rs_insert     rs_sqlserver_function_class

    --- Begin FString Text ---
 --------------------------------------------------------
     *** System-Supplied Transact-SQL Statement ***
       --- End FString Text ---
```

Usage

- If you do not supply a *function_name* parameter, rs_helpfstring displays all function strings defined for all functions of the replication definition.
- If you supply a *function_name* string, rs_helpfstring displays function strings that match *function_name*, such as rs_insert, rs_delete, rs_update, and rs_select, or a user-defined function.
- System-generated default function strings have no function string text stored in the RSSD. For these functions strings, rs_helpfstring displays the message "System-Supplied Transact-SQL Statement."

rs_helpfunc

Description

Displays information about functions available for a Replication Server or for a particular replication definition.

Syntax

rs_helpfunc [*replication_definition* [, *function_name*]]

Parameters

replication_definition
The replication definition for which you want function information.

function_name
A string of characters that corresponds to a function name. The string must match an entire function name or the first part of a name.

Examples

Example 1 Displays all available functions, replication definitions, and primary Replication Servers. The class scope of each function is also displayed.

```
rs_helpfunc
```

Example 2 Displays function information, including function names, parameters, and datatypes, for all functions of the replication definition authors_rep.

```
rs_helpfunc authors_rep


Functions and Parameters for Replication Definition:
                    'authors_rep'
System Function Names
 ---------------------
 rs_insert
 rs_delete
 rs_update
 rs_select
 rs_select_with_lock

Parameter(s)     Datatype  Length
 --------------- -------- ------
 @state           char          2
 @postalcode      char         10
 @au_id           varchar      11
 @phone           char         12
 @country         varchar      12
 @city            varchar      20
 @au_fname        varchar      20
 @address         varchar      40
 @au_lname        varchar      40
```

Example 3 Displays parameters and datatypes for the rs_insert function of the replication definition authors_rep.

```
rs_helpfunc authors_rep, rs_insert
```

Usage

- If you do not specify any parameters, rs_helpfunc lists all functions defined in the Replication Server.
- If you supply a *replication_definition* name, only the functions defined for that replication definition are listed. If you also supply a *function_name* string, rs_helpfunc displays functions whose names match *function_name*.
- rs_helpfunc notifies you if it detects duplicate user-defined functions that may interfere with asynchronous transactions.

rs_helppartition

Description
Displays information about Replication Server partitions.

Syntax
rs_helppartition [*partition_name*]

Parameters
partition_name
A string of characters that corresponds to a partition name. The string must match an entire partition name or the first part of a name.

Examples
Example 1 Displays summary information about all available database partitions for the Replication Server.

```
rs_helppartition
```

```
Displaying all partitions known to 'TOKYO_RS'.
Logical Name                   Size (MB)   Segments Allocated (MB)
------------------------------ ----------- -----------------------
partition_1                             20                       3
```

Example 2 Displays detailed information about the partition named partition_1.

```
rs_helppartition partition_1
```

```
Information for stable device: 'partition_1' on 'TOKYO_RS'.
This device is active.
Physical Name                                       Partition ID
--------------------------------------------------- ------------
/remote/tyrell2/app/dev/tokyo_rs_p1.dat                      101
Partition Size (MB)  Segments Allocated (MB)
-------------------- -------------------------
20                   5
Inbound Database Queue(s) on this partition:
Connection Name                                     Number of Segments
---------------------------------------------------- ------------------
LDS.pubs2                                                          1
TOKYO_DS.TOKYO_RSSD                                                1
Outbound Database Queue(s) on this partition:
Connection Name                                     Number of Segments
---------------------------------------------------- ------------------
LDS.pubs2                                                           1
TOKYO_DS.TOKYO_RSSD                                                 1
Outbound Replication Server Queue(s) on this partition:
Connection Name                                     Number of Segments
---------------------------------------------------- ------------------
SYDNEY_RS                                                   1
```

Usage
- If you do not specify any parameters, rs_helppartition lists summary information about all of the Replication Server's partitions.

- If you supply a *partition_name* string, rs_helppartition displays information about any partition whose name matches *partition_name*.
- If the *partition_name* string exactly matches a partition name, detailed information about the partition displays, including logical and physical name, total size, number of 1MB segments allocated from each partition, and queues on the partition.
- If the *partition_name* string does not exactly match a partition name, summary information displays for any partitions whose names match *partition_name* or for all known partitions.

rs_helppub

Description	Displays information about publications.
Syntax	rs_helppub [*publication_name*, *primary_dataserver*, *primary_db*, *article_name*]

Examples

Example 1

```
rs_helppub
```

```
 Publication Name            PRS              Primary DS.DB
 ----------------            -------          -------------
 funcpub                     prim_rs          P_DS.pdb1
 pub1                        prim_rs          P_DS.pdb1
 pub2                        prim_rs          P_DS.pdb1

 Num Articles      Status     Request Date
 ------------      -------    -------------------
 3                 Valid      Mar 23 1998 11:51AM
 7                 Valid      Mar 24 1998 10:41AM
 3                 Valid      Mar 24 1998 11:50AM

(return status = 0)
```

Example 2

```
rs_helppub funcpub:
```

```
Publication Name        PRS             Primary DS.DB
----------------        -------         --------------
funcpub                 prim_rs         P_DS.pdb1

Num Articles           Status           Request Date
------------           ------           -------------------
3                      Valid            Mar 23 1998 11:51AM

Article Name                       Replication Definition Type
------------                       ----------------------------
authors                            authors
authors                            authors
publishers                         publishers

Primary Object Name   Replicate Object Name   Request Date
------------------    ---------------------   -------------------
many_rows_data        many_rows_data          Mar 23 1998 10:01AM
                                              Mar 23 1998 11:51AM

Sub Name     Replicate DS.DB  Owner      Req. Date
```

```
----------   ---------------  -----      -------------------------
funcsub1     R_DS.rdb1         sa        Mar 24 1998 11:12AM

(return status = 0)
```

Example 3

```
rs_helppub funcpub, P_DS, pdb1, publishers:
```

```
Article Name  Publication Name    Replication Definition
------------  ----------------    ------------------------
publishers    funcpub             publishers

Primary Object Name           Replicate Object Name
--------------------          ---------------------
publishers                     publishers

Type      Request Date         Status
----      --------------       ----------
Table     Mar 23 1998 11:51AM   Valid

Where clauses
-----------------------------------------------------------------

  where
  pub_id = "0736"

Sub. Name       Replicate DS.DB   Owner      Req Date
----------      ---------------   -----      -------------------
funcsub1        R_DS.rdb1         sa         Mar 24 1998 11:12AM

(return status = 0)
```

Usage

- If rs_helppub is executed at the primary site, information displays for all of the publications created at that site.
- If rs_helppub is executed at the replicate site, information is displayed only for publications for which subscriptions have been created at that site.
- Use rs_helppubsub to display information about subscriptions to publications or articles.
- Use check_subscription to get the most accurate report of subscription status.

See also

rs_helppubsub

rs_helppubsub

Description

Displays information about publication subscriptions and article subscriptions.

Syntax

rs_helppubsub *subscription_name*, *publication_name*, *primary_dataserver*, *primary_db*, *replicate_dataserver*, *replicate_db*

Examples

Example 1 Lists all publication subscriptions known at this site:

```
rs_helppubsub
```

```
Subscription Name         Publication Name
------------------        ----------------
funcsub1                  funcpub

Primary DS.DB          Replicate DS.DB        PRS Status     RRS Status
--------------         ---------------        ----------     ----------
P_DS.pdb1              R_DS.rdb1              Unknown        Valid

Owner     Request Date
------    -------------------
sa        Mar 24 1998 11:12AM

Subscription Name                Article Name
------------------               -------------
funcsub1                         authors

Replication Definition         PRS Status    RRS Status
----------------------         ----------    ----------
authors                        Unknown       Valid

Request Date             Autocorrection
--------------------     --------------
Mar 24 1998 11:11AM      off
(return status = 0)
```

Example 2 Lists all publication subscriptions named *sub*.

```
rs_helppubsub sub
```

Example 3 Lists all publication subscriptions named *sub* for publications named *pub*.

```
rs_helppubsub sub, pub
```

Example 4 Lists all subscriptions named *sub* for the specified publication.

```
rs_helppubsub sub, pub, primary_dataserver, primary_db
```

Example 5 Lists the publication subscription and the article subscriptions in the group.

```
rs_helppubsub sub, pub, primary_dataserver, primary_db,
replicate_dataserver, replicate_db
```

```
Subscription Name     Publication Name     Primary DS.DB
 -----------------    ----------------     ----------------
 sub                  pub                  ost_cardhu_2.pdb1

 Replicate DS.DB      PRS Status     RRS Status     Owner
 -----------------    ----------     ----------     ---------
 ost_cardhu_2.rdb1    Unknown        Valid          rdb1_owner

 Request Date         Subscription Name     Article Name
 ----------------     -----------------     ------------
 February 25 1998     sub                   article1
                                            article2
                      sub                   article3
                      sub                   article4
                      sub                   article5

 PRS Status   RRS Status   Request Date   Replication Definition
 ----------   ----------   ------------   ----------------------
 Unknown      VALID        Feb 25, 1998   repdef1
                                          repdef2
 Unknown      VALID        Feb 25, 1998   repdef3
 Unknown      VALID        Feb 25, 1998   repdef4
 Unknown      VALID        Feb 25, 1998   repdef5

 Autocorrection   Subscribe to Truncate Table
 --------------   ---------------------------
 on               off
 off              on
 off              off
 off              off
```

Usage

- rs_helppub Use to determine all subscriptions for an article or a publication.
- Use check_subscription to get the most accurate report of subscription status.

See also

rs_helppub

rs_helprep

Description
Displays information about replication definitions.

Syntax
rs_helprep [*replication_definition*]

Parameters
replication_definition
A string of characters that corresponds to a replication definition name. The string must match an entire replication definition name or the first part of a name.

Examples

```
rs_helprep
```

Rep def	**PRS**	**Primary DS.DB**	**Primary table**	**Replicate table**	**Type**
authors	cardhu_11	cardhu_10.pdb1	authors	ling.authors_r1	Tbl
authors1	cardhu_11	cardhu_10.pdb1	authors	authors_r2	Tbl
discounts	cardhu_11	cardhu_10.pdb1	discounts	discounts	Tbl
publishers	cardhu_11	cardhu_10.pdb1	publishers	ling.publishers_r1	Tbl
publishers1	cardhu_11	cardhu_10.pdb1	publishers	publishers_r2	Tbl
roysched	cardhu_11	cardhu_10.pdb1	roysched	roysched	Tbl
rs_classes	cardhu_11	cardhu_10.emb	rs_classes	Tbl	
rs_columns	cardhu_11	cardhu_10.emb	rs_columns	Tbl	
rs_databases	cardhu_11	cardhu_10.emb	rs_databases	Tbl	
rs_erroractions	cardhu_11	cardhu_10.emb	rs_erroractions	Tbl	
rs_funcstrings	cardhu_11	cardhu_10.emb	rs_functstrings	Tbl	
rs_functions	cardhu_11	cardhu_10.emb	rs_functions	Tbl	
rs_objects	cardhu_11	cardhu_10.emb	rs_objects	Tbl	
rs_routes	cardhu_11	cardhu_10.emb	rs_routes	Tbl	
rs_systext	cardhu_11	cardhu_10.emb	rs_systext	Tbl	

```
rs_helprep authors_rep
```

Rep. Def. Name	**PRS**	**Type**	**Creation date**
authors	cardhu_11	Tbl	Oct 2, 1997 1:48PM
PDS.DB	*Primary Owner*		*Primary Table*
cardhu_10.db1			authors
Replicate Owner	*Replicate Table*		*Replicate Table*
ling	authors_r1		authors_r1

Rep. Def. Name	PRS	Type	Creation date
Send Min Cols.	*Used by Standby*		*Min Vers*
No	Yes		1000

Col. name	Rep. col. name	Datatype	Len.	Pri. col.	Searchable
au_id	au_id	varchar	11	1	1
au_lname	au_lname	varchar	40	0	1
au_fname	au_fname	varchar	20	0	0
phone	phone	char	12	0	0
address	address	varchar	40	0	0
city	city	varchar	20	0	0
state	state	char	2	0	0
zip	zip	char	12	0	0
contract	contract	bit	1	0	0

Function name	FString class	FString source	FString name
rs_delete	rs_sqlserver_function_class	Class Default	rs_delete
rs_insert	rs_sqlserver_function_class	Class Default	rs_insert
rs_select	rs_sqlserver_function_class	Class Default	rs_select
rs_select_with_lock	rs_sqlserver_function_class	Class Default	rs_select_with_lock
rs_truncate	rs_sqlserver_function_class	Class Default	rs_truncate
rs_update	rs_sqlserver_function_class	Class Default	rs_update

```
Subscriptions known at this Site 'cardhu_11'.
```

Subscription name	Replicate DS.DB	Owner	Creation date

(return status = 0)

Usage

- Unless you enter parameters, rs_helprep lists summary information for all replication definitions in the Replication Server.
- If you supply a *replication_definition* string, rs_helprep displays information about any replication definition whose name matches *replication_definition*.

- If the *replication_definition* string matches exactly one replication definition name, detailed information about that replication definition displays. Information includes the primary Replication Server, data server and database, replication definition columns, functions defined for the replication definition, and subscriptions for the replication definition known by the Replication Server.
- The detailed information displayed is slightly different for table replication definitions, function replication definitions, and system table replication definitions.
- If the *replication_definition* string does not match exactly one replication definition name, summary information is displayed for any replication definitions that match *replication_definition*.
- rs_helprep does not display database replication definition. Use rs_helpdbrep to display database replication definition.

rs_helprepdb

Description

Displays information about databases with subscriptions for replication definitions in the current Replication Server.

Syntax

rs_helprepdb [*data_server*, *database*]

Parameters

data_server
The data server with the database whose information you want to display.

database
The database whose information you want to display.

Examples

Example 1 Displays information about all databases with subscriptions for replication definitions in the current Replication Server.

```
    rs_helprepdb

dsname           dbname        dbid        controlling_prs
 ---------------- ------------- -----------  ----------------
 SYDNEY_DS         SYDNEY_RSSD    102          SYNDEY_RS
```

Example 2 Displays information about the specified data server and database.

```
    rs_helprepdb SYDNEY_DS, pubs2

dsname           dbname        dbid        controlling_prs
 ---------------- ------------- ----------  ----------------
 SYDNEY_DS         pubs2          104          SYDNEY_RS
```

Usage

- Execute rs_helprepdb in the RSSD for the primary Replication Server.
- Unless you specify *data_server* and *database* parameters, rs_helprepdb lists all databases with subscriptions for any of the Replication Server's replication definitions. The database ID and managing Replication Server display for each data server and database.
- If you supply the *data_server* and *database* parameters, rs_helprepdb displays information about the specified database only.

rs_helproute

Description

Provides status information about routes.

Syntax

rs_helproute [*replication_server*]

Parameters

replication_server
The name of a Replication Server for which you want route status information.

Examples

The route from TOKYO_RS to SYDNEY_RS is currently active.

```
rs_helproute

route                          route_status
-------------------------      ------------
TOKYO_RS -----> SYDNEY_RS       Active
```

Usage

- Unless you specify the *replication_server* parameter, rs_helproute displays information for all the routes known to the current Replication Server.
- If you supply a *replication_server*, information displays only for routes to and from that Replication Server.
- Replication Server uses a defined protocol to create and drop a route between the source and destination Replication Servers. During this protocol, the route goes through various states. rs_helproute, executed on the RSSD at the source or destination Replication Server, shows the current state of the protocol.
- For each route, rs_helproute returns two types of information:
  - Route status

    Status reflects the state of the route protocol. The information for each route depends on where you execute rs_helproute—at the route's source or destination.
  - List of system table subscriptions

    If you are creating a route, information is displayed about system table subscriptions that are being created. If you are dropping a route, this list tells you which system table subscriptions are being dropped.

    Routing protocols usually process system table subscriptions. This information helps you determine which subscriptions prevent you from proceeding to the next step in the protocol. If no system table subscriptions are listed, the protocol is currently not having problems with system table subscriptions.

Incomplete materialization or dematerialization of system table subscriptions is a common problem. If you notice any problems while creating, dropping, or altering routes, examine rs_helproute output for information about subscription status.

rs_helpsub

Description

Displays information about subscriptions.

Syntax

rs_helpsub
 [*subscription_name* [, *replication_definition*
  [, *data_server*, *database*]]]

Parameters

subscription_name
A string of characters that corresponds to a subscription name. The string must match an entire subscription name or the first part of a name.

replication_definition
The replication definition subscribed to.

data_server
The data server with the database containing the subscription's data.

database
The database containing the subscription's data.

Examples

Example 1 Displays summary information about all available subscriptions. The "Unknown" status in the last column reflects the fact that the current Replication Server has no knowledge of the subscription status at the listed Replication Server (the primary Replication Server).

```
rs_helpsub
```

```
** This Site is 'SYDNEY_RS' **
                                                        Status at
Subscription Name Rep. Def. Name  Replicate DS.DB  A/C RRS      PRS
----------------- --------------- ---------------- --- -------- ------
authors_sub       authors_rep     SYDNEY_DS.pubs2  0   Defined  Unknown
titles_sub        titles_rep      SYDNEY_DS.pubs2  0   Valid    Unknown
```

Example 2 Displays detailed information about the authors_sub subscription.

```
rs_helpsub authors_sub
```

```
Subscription Name Rep. Def. Name  Replicate DS.DB  A/C RRS      PRS
----------------- --------------- ---------------- --- -------- ------
authors_sub       authors_rep     SYDNEY_DS.pubs2  0   Defined  Unknown

Owner                 Creation Date
--------------------  -------------
sa                    Sep 22 1995

Subscription  Text
----------------------------------------------------------------------
```

```
create subscription authors_sub
  for authors_rep
  with replicate at SYDNEY_DS.pubs2
  where
  state = "CA"
```

Usage

- If you do not specify any parameters, rs_helpsub lists summary information about all subscriptions defined in the Replication Server. Information include replication definitions, replicate data server and database, autocorrection status, and subscription materialization status at the replicate and primary Replication Server.
- If you supply a *subscription_name* string, rs_helpsub displays information about any subscription whose name matches *subscription_name*.
- If the *subscription_name* string matches exactly one subscription name, the owner, creation date, and text of the subscription also display.
- If the *subscription_name* string does not match exactly one subscription name, summary information displays for any subscriptions whose names match *subscription_name*.
- If you also supply a *replication_definition*, rs_helpsub displays information only for subscriptions to that replication definition.
- rs_helpsub does not display subscription replication definition. Use rs_helpdbsub to display subscription replication definition.

rs_helpuser

Description	Displays information about user login names known to a Replication Server.
Syntax	rs_helpuser [*user*]
Parameters	*user* The user login name about which you want information.

Examples

Example 1 Displays information about all users.

```
rs_helpuser

                       Users and Privileges Known at Site repl_rs
Primary Users
User Name              Permission(s) Name
---------------------- ------------------------------
TOKYO_RS_id_user       no grants
sa                     sa
TOKYO_RS_ra            connect source
TOKYO_RS_rsi           connect source
repuser                create object
TOKYO_RSSD_prim        connect source, primary subscr

Maintenance Users
User name               Destination DS.DB
----------------------  ----------------------------------------------
TOKYO_RSSD_maint        TOKYO_DS.TOKYO_RSSD
pubs2_maint             TOKYO_DS.pubs2
pubs2_maint             SYDNEY_DS.pubs2sb
```

Example 2 Displays information about the pubs2_maint user.

```
rs_helpuser pubs2_maint

                       Users and Privileges Known at Site TOKYO_RS
Primary User(s)
User Name              Permission Name
---------------------- ------------------------------
pubs2_maint            TOKYO_DS.pubs2
pubs2_maint            SYDNEY_DS.pubs2sb
```

Usage

- Unless you enter parameters, rs_helpuser displays information about all user login names known to the current Replication Server.
- If you supply a *user* login name parameter, rs_helpuser displays information about that user login name only.

rs_helpreptable

Description

Displays information about replication definitions created against a primary table.

Syntax

rs_helpreptable *database*, [*owner*,] *table*

Parameters

database
The database where the table is created.

owner
The owner of the table.

table
The name of the table.

Examples

```
rs_helpreptable pdb1, authors
```

Replication definition name	Primary owner	Primary table	Primary owner	Replicate table	Used standby	Min vers
authors		authors	ling	authors_r1	Yes	1000
authors1		authors		authors_r2	No	1000

Usage

- Only user-defined table replication definitions are displayed.

rs_init_erroractions

Description

Initializes a new error class.

Syntax

rs_init_erroractions *new_error_class*, *template_class*

Parameters

new_error_class
The name of the new error class you have created.

template_class
The name of the error class that you want to serve as a template for the new error class.

Examples

Creates the error class new_class, based on the template error class, rs_sqlserver_error_class.

```
rs_init_erroractions new_class,
rs_sqlserver_error_class
```

Usage

- The template error class may be a user-defined error class or a system-provided error class such as rs_sqlserver_error_class.
- Use the create error class command to create the new error class in the primary Replication Server for that error class. Then use rs_init_erroractions to initialize the class.

See also

create error class

rs_ticket

Description

A stored procedure in the primary database that works with replicate database stored procedure rs_ticket_report 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.

Syntax

rs_ticket *h1* [, *h2* [, *h3* [, *h4*]]]

Parameters

h1 [, *h2* [, *h3* [, *h4*]]]
Short varchar strings. Header information.

Examples

Example 1 Executes rs_ticket at regular intervals:

```
Exec rs_ticket 'heartbeat'
```

Example 2 To measure performance, execute the following from the primary database:

```
Exec rs_ticket 'start'
Execute replication benchmarks
Exec rs_ticket 'stop'
```

Usage

- The rs_ticket stored procedure executes the following command:

  ```
  rs_marker 'rs_ticket rs_ticket_param'
  ```

  To avoid issuing wrongly formatted rs_marker and to enforce the *rs_ticket_param* standard, you should invoke rs_ticket instead of rs_marker. If you call rs_marker directly and form an incorrect rs_marker subcommand, the Replication Server refuses the rs_marker and shuts down the RepAgent connection. In this case, you must skip rs_marker from the transaction log, which may cause data loss.

- The Replication Server EXEC, DIST, and DSI modules parse and process rs_ticket subcommand:

  - When EXEC processes rs_ticket, it appends a timestamp, and then the total bytes received from RepAgent after *rs_ticket_param*. An EXEC timestamp takes the form "EXEC(spid)=hh:mm:ss.ddd". The byte information is "B(spid)=ddd". EXEC writes rs_ticket back to inbound queue.

  - When DIST processes rs_ticket, it appends another timestamp to *rs_ticket_param*. A DIST timestamp takes the form "DIST(spid)=hh:mm:ss.ddd".

    - When DSI processes rs_ticket, it appends yet another timestamp to *rs_ticket_param*. A DSI timestamp takes the form "DSI(spid)=hh:mm:ss.ddd".

- There are no subscriptions for rs_ticket. DIST does not send rs_ticket to DSI unless there is at least one subscription from the replicate site.
- rs_ticket is lightweight and nonintrusive and can be used in test environments as well as production environments.
- rs_ticket lets you know, without quiescing the Replication Server, when the data has been completely flushed out of replication path.
- The movement of rs_ticket is tracked by the EXEC, DIST, and DSI threads through RSTicket counter. Each thread has one RSTicket counter which is increased by one whenever the corresponding thread receives rs_ticket. This counter is never reset.

  You can monitor the module that rs_ticket has reached by sampling the RSTicket counters. RMS or other Replication Server monitoring tool uses these counters to produce EXEC, DIST, and DSI heartbeat.

  You can also monitor the health of the replication path by sending an rs_ticket at primary and checking the RSTicket counters. If RSTicket counter of a module is not increasing, it shows that replication path at this stage is broken.
- Use rs_ticket only when Replication Server is 15.0 or higher.

See also

rs_ticket_report stored procedure, rs_ticket_report function string

rs_ticket_report

Description

A stored procedure in the replicate database that users can customize to manipulate rs_ticket_param.

Syntax

rs_ticket_report *rs_ticket_param*

Parameters

rs_ticket_param
Contains timestamp and byte information for EXEC, DIST, and DSI transactions.

Examples

Customized rs_ticket_report stored procedure:

```
create procedure rs_ticket_report
@rs_ticket_param varchar(255)
as
begin
set nocount on

declare @new_cmd varchar(255),
        @c_time  datetime,
        @c_secs  numeric(6,3)

select @c_time = getdate()
select @c_secs = datepart( millisecond, @c_time)
select @c_secs = datepart( second, @c_time) + @c_secs/1000
select @new_cmd = @rs_ticket_param + ";RDB(" + db_name()
                   + ")=" + convert( varchar(2), datepart( hour, @c_time))
                   + ":" + convert( varchar(2), datepart minute, @c_time))
                   + ":" + convert( varchar(6), @c_secs)

insert ticket_history values (@c_time, @new_cmd)
end
go
```

Usage

- rs_ticket_report is invoked by the function string rs_ticket_report if dsi_rs_ticket_report is set to on.
- *rs_ticket_param* is written to Replication Server log when print_rs_ticket is on.
- Customize rs_ticket_report to parse and process *rs_ticket_param*.

See also

rs_ticket, rs_ticket_report function string

rs_zeroltm

Description

Resets the locator value for a database to zero (0). Use this stored procedure after you have used the Adaptive Server command dbcc settrunc to disable the secondary truncation point and truncate the logs, but before you restart Replication Server.

Syntax

rs_zeroltm *data_server*, *database*

Parameters

data_server
The data server with the database whose locator value you want to reset.

database
The database whose locator value you want to reset.

Examples

Resets the locator value to 0 for the TOKYO_DS data server and the pubs2 database.

```
rs_zeroltm TOKYO_DS, pubs2
```

Usage

- Use this command for RepAgent-enabled databases.
- Use dbcc settrunc to disable the secondary truncation point and truncate the log, before using rs_zeroltm.
- The locator value for a replicated database is maintained by the Replication Server and stored in the rs_locater table. Its value normally matches that of the secondary truncation point stored in the Adaptive Server.

  If the transaction log fills up, you may have to use the dbcc settrunc command to disable the secondary truncation point and truncate the log. dbcc settrunc resets the secondary truncation point, and the locator value and the secondary truncation point no longer match. Execute rs_zeroltm to bring the values back in sync: Setting the locator value to zero with rs_zeroltm tells Replication Server to get the new secondary truncation point from Adaptive Server and set the locator to that value.

See also

dbcc settrunc

CHAPTER 7 Executable Programs

This chapter contains reference pages for the Replication Server executable programs. These include Replication Server and the rs_subcmp procedure.

repserver

Description

The Replication Server executable program.

Syntax

repserver [-C *config_file*] [-i *id_server*] [-S *rs_name*] [-I *interfaces_file*] [-E *errorlog_file*] [-M] [-v] [-K *keytab_file*]

Parameters

-C *config_file*

Specifies the name and location of the Replication Server configuration file. The rs_init program creates a configuration file which, by default, is named *Rep_Server_name.cfg*, where *Rep_Server_name* is the name of the Replication Server. You can specify this file name by using the -C flag. If you do not use the -C flag, repserver looks for the configuration file named *config.rs* in the directory where you started the Replication Server.

-i *id_server*

Specifies the name of the ID Server for the replication system. The ID Server must be the first Replication Server started. It must be running and accessible before you can start a new Replication Server. The name of the ID Server is stored in the configuration file. Use the -i option to specify a different ID Server.

-S *rs_name*

The name to use for the current Replication Server. If network-based security and unified login are enabled, specifies the name of the principal user.

-I *interfaces_file*

Specifies the name and location of the interfaces file where the Replication Server is defined. The interfaces file must also have entries for the data servers and other Replication Servers that the current Replication Server communicates with. Interfaces files at replicate sites must have entries for the primary Replication Server and the primary data server. If you do not use the -I flag, Replication Server looks for the default interfaces file in the Sybase release directory.

Refer to the Replication Server installation and configuration guides for your platform for more information about the interfaces file, including the default interfaces file name for your platform.

-E *errorlog_file*

Specifies the name and location of the Replication Server error log file, into which repserver writes error messages. If you do not use the -E flag, the default error log file name and location is *repserver.log* in the directory where you started the Replication Server.

-M
Starts the Replication Server in standalone mode, which is used to initiate recovery actions. See the *Replication Server Administration Guide Volume 2* for more information about running Replication Server in standalone mode.

-v
Prints the version number of the Replication Server.

-K *keytab_file*
Should be used only with DCE network security. Specifies the name and location of the DCE keytab file that contains the security credential for the user logging into the server. Keytab files can be created with the DCE dcecp utility. See your DCE documentation for more information.

Note The -K *keytab_file* option is only applicable for Windows platforms.

Examples

Example 1 Starts the Replication Server named TOKYO_RS, using the configuration file TOKYO_RS.cfg.

```
repserver -STOKYO_RS -CTOKYO_RS.cfg
```

Example 2 Starts the Replication Server named SYDNEY_RS, using the configuration file SYDNEY_RS.cfg. TOKYO_RS is the ID Server for the replication system.

```
repserver -SSYDNEY_RS -CSYDNEY_RS.cfg -iTOKYO_RS
```

Example 3 Starts Replication Server and specifies an interfaces file, *my_newinterfaces*, that overrides a default interfaces file or LDAP directory service.

```
repserver -STOKYO_RS _CTOKYO_RS.cfg
    -I$SYBASE/SYBASE_RS/my_newinterfaces
```

Usage

- Use the repserver command to start the Replication Server executable program. Normally, you start Replication Server by executing the run file created by rs_init.
- On UNIX systems, this executable program is called repserver. On PC systems, the program is called repsrvr.
- The repserver executable program is located in the *bin* subdirectory of the Sybase release directory. Refer to the Replication Server installation and configuration guides for your platform for more information.
- The repserver command should be executed by the “sybase” user so that the Replication Server can access its disk partitions.

- The interfaces file must contain definitions of the other Replication Servers and data servers that the current Replication Server communicates with. Interfaces files at replicate sites must have entries for the primary Replication Server and the primary data server.
- If a password is stored in encrypted form, you cannot edit it directly by editing the Replication Server configuration file. To change an encrypted password in this file, use the rs_init program. Refer to the Replication Server installation and configuration guides for your platform for more information.
- The RSSD_primary_user and the RSSD_maint_user are automatically assigned to the rs_systabgroup group by rs_init at Replication Server configuration time. This enables these users to modify the system tables. You can add other user login names to this group with the Adaptive Server system procedure sp_changegroup. See the *Adaptive Server Enterprise System Administration Guide* for more information.
- If any of the network-based security parameters for the RSSD are present, the use_security_services parameter is set "on" and network-based security is initiated automatically.

Replication Server configuration file

The following table lists the parameters in the Replication Server configuration file.

Table 7-1: Replication Server configuration file parameters

Configuration parameter	**Description**
CONFIG_charset	The character set used to write the Replication Server configuration file. Use this parameter only if this character set differs from the Replication Server's character set. It can be any character set that is compatible with the Replication Server's character set.
erssd_backup_dir	ERSSD backup directory.
erssd_dbfile	ERSSD database file.
erssd_errorlog	ERSSD error log.
erssd_logmirror	ERSSD transaction log mirror file.
erssd_ping_cmd	Allows user to specify a different command to ping ERSSD. For debug purposes only.
erssd_port	ERSSD port number for network listener. The port number is obtained from the interface file.
erssd_release_dir	Allows user to specify a different release directory. For debug purposes only. The default is *$SYBASE/$SYBASE_REP/ASA9*.
erssd_ra_release_dir	Allows a user to specify a different release directory for ERSSD Replication Agent. For debug purposes only.

Configuration parameter	Description
erssd_ra_start_cmd	Allows user to specify a different command to start ERSSD Replication Agent. For debug purposes only.
erssd_start_cmd	Allows user to specify a different command to start ERSSD. For debug purposes only.
erssd_translog	ERSSD transaction log file.
ID_pw	The password for the ID Server user (*ID_user*).
ID_pw_enc	The encrypted password for the ID Server user (*ID_user*).
ID_server	The name of the Replication Server that is the designated ID Server for the replication system.
ID_user	The login name on the ID Server for other Replication Servers to use.
RS_charset	The character set for the Replication Server to use. You can specify any Sybase-supported character set. In setting up a replication system, it is highly recommended, though not required, that all servers at a given Replication Server site use the same character set. It is also recommended that all of the Replication Servers in your replication system use compatible character sets. Refer to the *Replication Server Design Guide* for details.
RS_language	The language used by the Replication Server to print its messages to the error log file and to its clients. You can specify any language to which the Replication Server has been localized that is compatible with the character set chosen.
RS_send_enc_pw	Ensures that all Replication Server client connections are made with encrypted passwords except for the first connection to the RSSD. Values are on and off. Default: off
RS_sortorder	The sort order that Replication Server uses. The sort order controls what rows of a table belong in a subscription that has a where clause involving character data. It also controls how identifiers you enter are recognized. You can specify any Sybase-supported sort order that is compatible with the character set chosen. All sort orders in your replication system should be the same.
RS_unicode_sort_order	The Unicode sort order Replication uses. You can specify any Sybase-supported Unicode sort order. Default: binary
RSSD_database	The name of the RSSD.
RSSD_embedded	Indicates whether RSSD is embedded or not.
RSSD_ha_failover	Specifies whether HA failover is allowed or not. Default: No.
RSSD_maint_pw	The password for the RSSD maintenance user.
RSSD_maint_pw_enc	The encrypted password for the RSSD maintenance user.

Configuration parameter	Description
RSSD_maint_user	The login name for the RSSD maintenance user. This login name is automatically assigned to the rs_systabgroup group, whose users can modify the system tables. You can add other user login names to this group with the Adaptive Server system procedure sp_changegroup. See the *Adaptive Server Enterprise System Administration Guide* for more information.
RSSD_msg_confidentiality	Specifies whether Replication Server sends and receives encrypted data. If set to “required”, outgoing and incoming data must be encrypted. If set to “not_required”, outgoing data is not encrypted and incoming data may be encrypted or not encrypted. This option is not implemented. Default: not_required
RSSD_msg_integrity	Specifies whether data are checked for tampering. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required
RSSD_msg_origin_check	Specifies whether the origin of data should be checked. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required
RSSD_msg_replay_detection	Specifies whether data should be checked to make sure they have not been read or intercepted. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required
RSSD_msg_sequence_check	Specifies whether data should be checked to make sure the sequence hasn’t changed. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required
RSSD_mutual_auth	Specifies whether the RSSD must provide proof of identity before Replication Server establishes a connection. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required
RSSD_primary_user	The login name for the RSSD primary user. rs_init automatically assigns this user to the rs_systabgroup group during installation. You can add other user login names to this group using the Adaptive Server system procedure sp_changegroup. See the *Adaptive Server Enterprise System Administration Guide* for more information.
RSSD_primary_pw	The password for the RSSD primary user.
RSSD_primary_pw_enc	The encrypted password for the RSSD primary user.
RSSD_sec_mechanism	The security mechanism Replication Server uses for initial contact with the RSSD at startup. Thereafter, network security information for contact with the RSSD is read from the *rs_config* file. This option is not implemented.
RSSD_server	The name of the Adaptive Server with the RSSD.
RS_ssl_identity	SSL identity file.

Configuration parameter	Description
RS_ssl_pw	Password for the SSL private key
RS_ssl_pw_enc	The encrypted password for the SSL private key.
RSSD_unified_login	Specifies whether Replication Server seeks to connect to the RSSD with a credential at startup. Thereafter, network security information for contact with the RSSD is read from the *rs_config* file. Valid entries are “required” and “not_required”. This option is not implemented. Default: not_required

trace	Turns on a Replication Server trace. You can use multiple instances of this parameter to set the different traces available. Spaces are not allowed. For example:
	
	`trace=DSI,DSI_BUF_DUMP`
	`trace=DIST,DIST_TRACE_COMMANDS`

trace_file	Indicates the name of the Replication Server log file.

rs_subcmp

Description

An executable program that compares the data of a replicated table to the primary version of the table. rs_subcmp also performs schema comparison between replicated and primary tables and between replicated and primary databases. These features aid in finding—and optionally reconciling—missing, orphaned, and inconsistent rows and schemas. On UNIX systems, this program is called rs_subcmp. On PC systems, the program is called subcmp.

The rs_subcmp program is located in the *bin* subdirectory of the Sybase release directory. See the Replication Server installation and configuration guides for your platform for more information.

For rs_subcmp to work, the *SYBASE* environment variable, and the library path environment variable must be set. If you use rs_subcmp for schema comparison, set the *DDLGENLOC* environment variable, and the *SYBROOT* environment variable. See the Usage section for instructions.

rs_subcmp is intended to reconcile Sybase databases only.

Syntax

rs_subcmp [-R | -r] [-v] [-V] [-z[1 | 2]]
 [-f *config_file*] [-F]
 -S *primary_ds* [-D *primary_db*]
 -s *replicate_ds* [-d *replicate_db*]
 -t *table_name* [-T *primary_table_name*]
 -c *select_command* [-C *primary_select_command*]
 -u *user* [-U *primary_user*]
 [-p *passwd*] [-P *primary_passwd*]
 [-B *primary_init_batch*]
 [-b *replicate_init_batch*]
 [-n *num_iterations*] [-w *wait_interval*]
 [-e *float_precision*] [-E *real_precision*]
 [-k *primary_key_column* [-k *primary_key_column*]...]
 [-i *identity_column*]
 [-l *text_image_column_name*
 [-l *text_image_column_name*]...]
 [-L *text_image_length_in_kilobytes*]
 [-N *text_image_column_name*
 [-N *text_image_column_name*]...]
 [-Z *language*]
 [-o *sort_order*]
 [-O *sort_order*]
 [-J *rs_subcmp_charset*]
 [-j *rep_charset*]
 [-a *replicate_column_name primary_column_name*
 [-a *replicate_column_name primary_column_name*]...]
 [-q *unicode_sort_order*]
 [-Q *unicode_sort_order*]
 [-x *schema_flag*]

[-X *filter_flag*]
[-I *interface_file*]

Parameters

-R
Reconciles the replicate data with the primary data, making a final verification of data inconsistencies at the primary database. rs_subcmp inserts, deletes, and updates rows at the replicate database so that the replicate data matches the primary data.

-r
Reconciles the replicate data with the primary data, without making a final verification of data inconsistencies at the primary database, as -R does. rs_subcmp inserts, deletes, and updates rows at the replicate database so that the replicate data matches the primary data.

-v
Prints version information.

-V
(Visual) prints the results of the comparison on the display (standard output). If you do not use the -V flag, rs_subcmp does not report differences between rows. Values of text, unitext, or image data are not printed. Instead, rs_subcmp reports whether the inconsistency is in the text, unitext, or image columns or in the columns of other datatypes.

-z
Enables trace. -z1, the default, provides basic trace information, such as comparisons of column headings. -z1 also prints information about numeric precision differences. -z2 provides trace information on comparisons of all rows and commands.

-f *config_file*
Specifies the name of the configuration file for rs_subcmp.

-F
Displays the format (syntax) to use for the *config_file*. A configuration file must use the syntax displayed with the -F option, and must contain all required syntax parameters.

-S *primary_ds*
The name of the data server with the primary data for the subscription.

-D *primary_db*
The name of the database where the primary data for the subscription is stored.

-s *replicate_ds*
The name of the data server with the replicate copy of the data.

-d *replicate_db*
The name of the database with the replicate copy of the data.

-t *table_name*
The name of the table in the primary and replicate databases with the data to be compared. If the name is different in the databases, use the -T option to specify the name of the table in the primary database. You can include table owner name information here.

-T *primary_table_name*
The name of the table in the primary database. Use this option when the table name is different in the primary and replicate databases. You can include table owner name information here.

-c *select_command*
A select command that retrieves the subscription's data from both the primary and replicate copies of the data. Use -C to specify a different command for the primary data. select commands must order rows based on the primary key.

You can include columns with text, unitext, or image datatypes in the select command, with the following requirements:

- Columns with text, unitext, or image datatypes cannot be primary key columns.
- You must place columns with text, unitext, or image datatypes at the end of the select list.
- By default, the replicate table does not allow null values for text or image columns. You must include the -N flag in the rs_subcmp executable to indicate that a null value is allowed in the text, unitext, or image column of the replicate table.

-C *primary_select_command*
A select command that retrieves the subscription's data from the primary copy of the data. Use this option and -c when you need a different select command for the primary and replicate databases. select commands must order rows based on the primary key.

-u *user*
The login name used to log into the primary and replicate data servers. If you need different login names, use the -U option to specify a different primary data server login name.

-U *primary_user*
The login name used to log into the primary data server. Use this option and the -u option when different login names are required for the primary and replicate data servers.

-p *passwd*
The password to use with the *user* login name and, if supplied, the *primary_user* login name. If you omit this option, rs_subcmp uses a null password. If you specify a different password for the *primary_user* login name, specify it with the -P option.

-P *primary_passwd*
The password to use with the *primary_user* login name.

-B *primary_init_batch*
A command batch to be executed when initially connecting to the primary database. The batch can be used for any purpose, such as to set the isolation level. The batch is run after rs_subcmp logs into the primary database.

-b *replicate_init_batch*
A command batch to be executed when initially connecting to the replicate database. The batch can be used for any purpose, such as to turn off triggers when running rs_subcmp in a warm standby application, or to set the isolation level. The batch is run after rs_subcmp logs into the replicate database.

-n *num_iterations*
The number of times that rs_subcmp examines the inconsistent rows it finds. The default is 10 iterations. The first iteration may find many inconsistencies due to normal time lag in replication. Additional iterations allow rs_subcmp to distinguish true inconsistencies from the inconsistent rows that are corrected through normal replication activity.

-w *wait_interval*
The number of seconds rs_subcmp waits before beginning another iteration. The default is 5 seconds.

-e *float_precision*
Sets the number of decimal places in exponential notation that floating point values are expected to agree. By default, this is set to the maximum precision supported by the platform.

-E *real_precision*
Sets the number of decimal places in exponential notation that real values are expected to agree. By default, this is set to the maximum precision supported by the platform.

-k *primary_key_column*
: A column name that is part of the primary key for the table. The primary key must be unique and it cannot be a text, unitext, or image column. Use the -k option for each column in the primary key. If the primary and replicate column names are different, the name specified here is the replicate column name.

-i *identity_column*
: The name of the xidentity column in the replicate table.

-l *text_image_column_name*
: Turns off logging of updates to a replicate text, unitext, or image column. By default, text, unitext, or image column updates are logged.

-L *text_image_length*
: Sets the longest value the data server returns for text, unitext, or image columns. The default value is 2048 KB.

-N *text_image_column_name*
: Indicates that a null value is allowed in the text, unitext, or image column of the replicate table. By default, the replicate table does not allow null values for text, unitext, or image columns.

-Z *language*
: The name of the language in which rs_subcmp generates error and informational messages. If not specified, it uses the language specified in the "default" locale entry for your platform.

-o *sort_order*
: The name of the sort order used in your replication system. rs_subcmp uses this information to compare primary key columns.

-O *sort_order*
: The name of the sort order used in your replication system. rs_subcmp uses this information to compare all columns.

-J *rs_subcmp_charset*
: The name of the character set used by rs_subcmp error and informational messages and in all configuration parameters and command line options. If you do not specify *rs_subcmp_charset*, it is set to the character set specified in the "default" locale entry for your platform.

-j *rep_charset*
: The name of the character set used by the replicate data server. The rs_subcmp program uses this character set when comparing and reconciling the replicate and primary versions of a table. If you do not specify a *rep_charset,* it is set to the *rs_subcmp_charset* character set.

-a *replicate_column_name primary_column_name*
Specifies the primary column name associated with a replicate column. Use this option if a replicate column name is different from that of the primary column.

Note When you use the -a option, the replicate column name must come before the associated primary column name.

-q *unicode_sort_order*
Specifies the Unicode sort order rs_subcmp uses to compare Unicode primary key columns.

-Q *unicode_sort_order*
Specifies the Unicode sort order rs_subcmp uses to compare all Unicode columns.

-x *schema_flag*
Specifies the rs_subcmp comparison type. The possible values of the *schema_flag* are:

- 0 – data comparison. This is the default value.
- 1 – database schema comparison between two databases.
- 2 – table schema comparison between two tables.

-X *filter*
Specifies the schema types and subtypes included or excluded from the comparison. If the value starts with "+", only the schema types are selected for comparison, and the subschema types are ignored. Otherwise, the schema types and subschema types are both unselected and not used for comparison. For a list of schema types and schema subtypes supported by rs_subcmp, see Table 7-4 and Table 7-5.

-I *interface_file*
Specifies the interface file location. For more information on the interface file, see the Replication Server configuration guides for your platform.

-g
Creates reconciliation file for inconsistent data.

-h
Performs fast comparison.

-H *normalization_option*
Indicates how to normalize the data when performing fast comparison. For a list of normalization options supported by rs_subcmp, see Table 7-6.

Examples

Example 1 Starts rs_subcmp using a configuration file called *titleauthor.cfg*.

```
rs_subcmp -ftitleauthor.cfg
```

The configuration file consists of the following:

```
# titleauthor.cfg - Reconcile
# SYDNEY_DS.pubs2.dbo.titleauthor with
# TOKYO_DS.pubs2.dbo.titleauthor.
#
PDS       = TOKYO_DS
RDS       = SYDNEY_DS
PDB       = pubs2
RDB       = pubs2
PTABLE    = titleauthor
RTABLE    = titleauthor
PSELECT   = select au_id, title_id, au_ord,\ royaltyper
    from titleauthor order by au_id,\ title_id
RSELECT   = select au_id, title_id, au_ord,\ royaltyper
    from titleauthor order by au_id,\ title_id
PUSER     = repuser
RUSER     = repuser
PPWD      = piglet
RPWD      = piglet
KEY       = au_id
KEY       = title_id
RECONCILE = Y
VISUAL    = Y
NUM_TRIES = 3
WAIT      = 10
```

rs_subcmp compares the primary and replicate tables called titleauthor and generates the following output:

```
$SYBASE/bin/rs_subcmp -f ttl_au.cmp
INCONSISTENT ROWS:

_________Replicate row________
au_id        title_id  au_ord  royaltyper
-------------------------------------------
672-71-3249  TC7777    1       40

__________Primary row_________
au_id        title_id  au_ord  royaltyper
-------------------------------------------
672-71-3249  TC7777    1       50
```

Example 2 Starts rs_subcmp using a configuration file called *subcmp.cfg*. Command line flags override the configuration file settings, to reconcile differences in the primary and replicate versions of the authors table, performing a final verification.

```
rs_subcmp -R -fsubcmp.cfg -STOKYO_DS -Dpubs2 \
 -sSYDNEY_DS -dpubs2 -tauthors
```

The primary data server and database are TOKYO_DS and pubs2. The replicate data server and database are SYDNEY_DS and pubs2.

Example 3 Compares all schemas between two databases using the *config.cfg* file:

```
rs_subcmp -f config.cfg
```

The configuration file contains:

```
PDS = PASE
RDS = R2ASE
PDB = pubs2
PTABLE = authors
RTABLE = authors
PUSER = sa
RUSER = sa
PPWD =
RPWD =
SCHEMAFLAG = 1
```

Example 4 Compares schema between two databases without a configuration file:

```
rs_subcmp -Spds -srds -Dpdb -drdb -Usa -usa -Psa_pwd
          -psa_pwd -x1
```

Example 5 Compares schema of two databases excluding index, trigger, and datatype:

```
rs_subcmp -Spds -srds -Dpdb -drdb -Usa -usa -Psa_pwd
          -psa_pwd -x1 -XitD
```

Example 6 Compares all table schemas and user schemas:

```
rs_subcmp -Spds -srds -Dpdb -drdb -Usa -usa -Psa_pwd
          -psa_pwd -x1 -X+TU
```

Usage

- Run rs_subcmp when primary changes do not occur.
- The *SYBASE* environment variable, and the library path environment variable must be set for rs_subcmp to work.

  Set the *SYBASE* environment variable to the Sybase release directory.

Set the library path variable to *$SYBASE/$SYBASE_OCS/lib* (UNIX) or *%SYBASE%\%SYBASE_OCS%\lib* (Windows):

  - For Solaris and Linux, the library path variable is LD_LIBRARY_PATH.
  - For HP, the library path variable is SHLIB_PATH.
  - For RS6000, the library path variable is LIBPATH.
  - For Windows, the library path variable is PATH.

- The *DDLGENLOC* and *SYBROOT* environment variables must be set for the rs_subcmp schema comparison feature to work.

  Set *DDLGENLOC* environment variable to the location of the ddlgen executable file. If the environment variable is not set, then rs_subcmp will set *DDLGENLOC* to *$SYBASE/ASEP/bin/ddlgen*.

  Set *SYBROOT* environment variable to the SYBASE environment variable.

- The following requirements apply to rs_subcmp:

  - If you provide a configuration file and also use command line options, the command line values override the values in the configuration file.
  - The lowercase options -d, -c, -u, -p, and -t provide values for both primary and replicated data. Use the uppercase options to override the values for primary data.
  - The only required uppercase option is -S.
  - The primary key specified with -k must be unique. If you do not specify any primary key columns with the -k option, all columns are considered to be part of the primary key.
  - Use a positive integer in -L to specify a new value, overwriting the default value of 26 KB, for the byte length of text and image columns:

    ```
    -L = <new_value>
    ```

    For instance, if you want text and image columns to be 65,536 bytes, enter:

    ```
    -L = <64>
    ```

- These options can be used to specify a non-default table owner or a different primary replicate table or column name:

  - For options -t, -T, -c, and -C, table owner information can be included (for example, ling.authors).

- Owner, table, and column names specified for the -c option should be those of the replicate table.
- Owner, table, and column names specified for the -C option should be those of the primary table.
- The column name specified for the -k option is the column name of the replicate table.

- rs_subcmp creates a report file after every schema comparison. The report file details the comparison result between two tables or two databases. The report file is named *reportPROCID.txt*. If inconsistencies exist, rs_subcmp creates a reconciliation script named *reconcilePROCID.sql*. The report file and the reconciliation script are saved in the same directory from which rs_subcmp executed.
- The reconciliation file's SQL statements cannot contain text, unitext, or image.
- rs_subcmp creates a reconciliation file if you specify the -g option. The file is named *reconcile_file_PROCID.sql* and is located at the current working directory.

Return codes

The following return codes can be returned by rs_subcmp:

Table 7-2: rs_subcmp return codes

Return code	Meaning
0	The replicated and primary tables are the same.
1	An error occurred while executing rs_subcmp.
2	The replicated and primary tables are different.

Configuration file

You can create a file containing rs_subcmp parameters and specify it on the command line using the -f flag. Each line in the configuration file consists of a parameter name, an equal sign (=), and a value.

The following table lists the parameters that can be used in the rs_subcmp configuration file and the corresponding command line option for each parameter.

Table 7-3: rs_subcmp configuration file parameters

Configuration parameter	Command-line option	Value
PDS	-S	Primary data server name
RDS	-s	Replicate data server name

Configuration parameter	Command-line option	Value
PDB	-D	Primary database name
RDB	-d	Replicate database name
PTABLE	-T	Primary table name
RTABLE	-t	Replicate table name
PUSER	-U	Primary user name
RUSER	-u	Replicate user name
PPWD	-P	Primary password
RPWD	-p	Replicate password
KEY	-k	Primary key element in replicate table
PINITBATCH	-B	Primary database connection initialization batch. Can span multiple lines if newline characters are preceded by a “\” (backslash). Up to 1024 characters per line and 64K characters total are allowed.
RINITBATCH	-b	Replicate database connection initialization batch. Can span multiple lines if newline characters are preceded by a “\” (backslash). Up to 1024 characters per line and 64K characters total are allowed.
PSELECT	-C	Primary select command. Can span multiple lines if newline characters are preceded by a “\” (backslash). Up to 1024 characters per line and 64K characters total are allowed.
RSELECT	-c	Replicate select command. Can span multiple lines if newline characters are preceded by a “\” (backslash). Up to 1024 characters per line and 64K characters total are allowed.
RECONCILE	-r	Reconcile differences (Y or N)
RECONCILE_CHECK	-R	Reconcile differences with primary verification (Y or N)
TRACE	-z	Enable trace with optional level (optional integer)
FPRECISION	-e	Expected floating point precision (integer—default is platform-dependent)
RPRECISION	-E	Expected real precision (integer—default is platform-dependent)
WAIT	-w	Seconds between comparisons (integer—default is 5 seconds)
NUM_TRIES	-n	Number of comparisons (integer—default is 10 iterations)
VISUAL	-V	Print results (Y or N)
IDENTITY	-i	identity column name in replicate table
TXT_IMG_LEN	-L	The longest value, in kilobytes, the data server returns for text, unitext, or image columns.
NO_LOG	-l	Do not log updates for this replicate text, unitext, or image column
NULLABLE	-N	The text, unitext, or image column in the replicate table accepts null values.
LANGUAGE	-Z	Language of rs_subcmp error and informational messages

Configuration parameter	Command-line option	Value
SORT_ORDER	-o	Use the specified sort order to compare primary key columns.
SORT_ORDER_ALL_COLS	-O	Use the specified sort order to compare all columns.
SCHARSET	-j	Character set of rs_subcmp
RCHARSET	-J	Character set of the replicate data server
REP_PRI_COLNAME	-a	Replicate-Primary column name pair
UNICODE_SORT_ORDER	-q	The Unicode sort order rs_subcmp uses to compare Unicode primary key columns.
UNICODE_SORT_ORDER_ ALL_COLS	-Q	The Unicode sort order rs_subcmp uses to compare all Unicode columns.
SCHEMAFLAG	-x	The rs_subcmp comparison type.
FILTER	-X	The filter used to indicate the schema and schema subtypes included or excluded in the schema comparison. See Table 7-4 and Table 7-5 for a list of schema types and schema subtypes supported by rs_subcmp.
IFILE	-I	The interface file location.
RECONCILE_FILE	-g	Indicates whether to create a reconciliation file or not. Values: • Y – create reconciliation file. • N – do not create reconciliation file. Default: N
FASTCMP	-h	Indicates whether to perform fast comparison or not. Values: • Y – perform fast comparison using compressed data. • N – perform normal comparison. Default: N
HASH_OPTION	-H	Indicates the normalization option used for fast comparison. If this parameter is not included in the configuration file, rs_subcmp normalizes the data using native byte order and character set. See Table 7-6 for a list of the normalization options supported by rs_subcmp.

Requirements for *select* commands

- The select commands specified by -c (RSELECT) and -C (PSELECT) must return columns with the same names and datatypes from both the primary and the replicate databases.

- You must have a clustered index on the primary key or an order by clause in the select command. select commands must order rows based on the primary key. If rs_subcmp does not receive rows in the correct order, it may delete rows in the replicate table.
- Do not select rs_address datatypes with the -c or -C options. If replicate tables contain columns using the rs_address datatype, the primary and replicate versions of these columns may not be identical. Replication Server filters out updates to these columns so as not to replicate them unnecessarily.

How *rs_subcmp* works

- rs_subcmp logs into the primary and replicate databases and executes the supplied select commands. It verifies that the commands return the same columns, based on the name and datatype of each column. If the returned columns match, rs_subcmp compares the primary and replicate rows and creates these lists:
  - Missing rows – rows at the primary, but not at the replicate
  - Orphaned rows – rows at the replicate, but not at the primary
  - Inconsistent rows – rows at the replicate and the primary with matching primary keys, but differences in other columns
- After the three lists are compiled rs_subcmp iterates for the specified number of times, checking:
  - If missing rows appear at the replicate
  - If orphaned rows disappear from the replicate
  - If inconsistent rows match
  - If the new replicate row value matches the primary row value from the previous iteration
- After the specified number of iterations, the contents of the three lists are printed to the standard output if you specified the -V option.

Reconciling inconsistencies

- rs_subcmp reconciles missing, orphaned, and inconsistent rows if you specify the -R or -r option.
- If you specify the -r option, rs_subcmp reconciles the primary and replicate copies. It passes the final lists and modifies the replicate table as follows:
  - Inserts rows remaining in the missing rows list

    - Deletes rows remaining in the orphaned rows list
    - Updates inconsistent rows to match the primary rows
- If you specify the -R option, rs_subcmp reconciles the replicate table to the primary version in the same way as with the -r option. However, before it inserts a missing row or deletes an orphaned row, it logs into the primary database and performs a select on the row to verify that:
    - The row still exists (in the case of a missing row in the replicate table), or
    - The row does not exist (in the case of an orphaned row in the replicate table).

Reconciling IDENTITY columns

- If the values in an identity column for a row are inconsistent, rs_subcmp reconciles them by deleting the row in the replicate database before inserting the row from the primary database.

Reconciling *text*, *unitext*, or *image* datatypes

- Unlike other datatypes, inconsistencies in text, unitext, or image values are not stored in a list. To reconcile a missing or inconsistent row that contains a text or image value, rs_subcmp logs back into the primary database and re-executes the select statement. If the inconsistent or missing row is found, rs_subcmp modifies the replicate table by updating or inserting the row. However, if the inconsistent or missing row is not found in the primary table, rs_subcmp takes the following actions:
    - For an inconsistent row, rs_subcmp deletes the row from the replicate table
    - For a missing row, rs_subcmp takes no action
- Using the Adaptive Server option set textsize as part of the select statement can limit the amount of text compared. For example, the following example shows the effect of setting the textsize to 10. The first select statement returns 30 characters of text:

```
set textsize 30 select * from zetext

a           b            c
--------    ---------    -----------
abba        apples       odd one here
beta        banana       rotten
caro        celery       not carrots
```

The next select statement sets the size of the text to 10:

```
1> set textsize 10 select * from zetext
2> go
a           b           c
--------    ---------   ------------
abba        apples      odd one
beta        banana      rotten
caro        celery      not carrots
---------------------------------------------------

(3 rows affected)
```

Using *rs_subcmp* in international environments

- rs_subcmp provides support for international environments with the -Z*language*, -o *sort_order*, -O *sort_order*, -q *unicode_sort_order*, -Q *unicode_sort_order*, -J *rs_subcmp_charset*, and -j *rep_charset* options.
- rs_subcmp performs character set conversion when comparing and reconciling the replicate and primary versions of a table. The method is similar to how Replication Server converts character sets, so you can expect to see similar results.

  For example, if the primary and replicate data server's character sets are incompatible, no conversion takes place. If the character sets are incompatible but a single character from the primary data server's character set has no representation in the replicate server's character set, the character is replaced with a "?" and processing continues.

- rs_subcmp uses the character set of the replicate data server in all operations involving user data. To specify the replicate data server's character set, use the -j command line option or the *RCHARSET* configuration file parameter.

  Note rs_subcmp does not have a parameter for the primary data server's character set because all data operations are done in the replicate data server's character set. The program depends on the primary data server to convert all character data to the replicate data server's character set. This is comparable to how Replication Server works during subscription materialization.

- You can also specify a character set for rs_subcmp if it is different from the replicate data server's character set. To do this, use the -J command line option or the *SCHARSET* configuration file parameter. When you specify a character set, rs_subcmp converts its string-type configuration parameters from the rs_subcmp character set to the replicate data server's character set.

Requirements for character sets and sort orders

- The following requirements apply for specifying character sets and sort orders in rs_subcmp:
  - All characters in object names (including servers, databases, tables, and column names) must be compatible with the *rs_subcmp_charset* and *rep_charset* character sets; otherwise rs_subcmp will fail to execute.
  - If the character sets of the replicate and primary data servers differ, the replicate data server's character set must be installed at the primary data server. This enables the primary data server to do character set translation.
  - If the replicate and primary data servers use different sort orders and the where clause of the select statement includes character or text datatypes, results may be confusing. To avoid confusion, run rs_subcmp first without the -r or -R (reconcile) options and with the -V (visual) option to see the potential effects on your data.

Using sort orders

- You can specify nonUnicode sort order in two ways: using the -o option or using the -O option.
- If you specify the -o option, rs_subcmp:

a. Performs a simple binary comparison of the primary key columns.
b. If the primary keys match, rs_subcmp performs a binary comparison of the remaining columns. If they don’t match, an inconsistent row is reported.
c. If the primary key columns do not match, rs_subcmp compares them using the specified sort order.
    - If the primary key columns don’t match, the row is reported missing or orphan.
    - If the primary key columns test equal using the sort order, the row is reported inconsistent.

- If you specify the -O option, rs_subcmp:
    - Performs a column comparison using the specified sort order for all columns of types char, varchar, and text.
    - Does not perform a binary comparison.
- If no sort order is specified, rs_subcmp performs a simple binary comparison on each column of the primary and replicate row.

Using Unicode sort orders

- You can specify Unicode sort order in two ways: using the -q option or using the -Q option.
- If you specify the -q option, rs_subcmp:

    a. Performs a simple binary comparison of the Unicode primary key columns.
    b. If the primary keys match, rs_subcmp performs a binary comparison of the remaining columns. If they don’t match, an inconsistent row is reported.
    c. If the primary key columns do not match, rs_subcmp compares them using the specified sort order.
        - If the Unicode primary key columns don’t match, the row is reported missing or orphan.
        - If the primary key columns test equal using the sort order, the row is reported inconsistent.

- If you specify the -Q option, rs_subcmp:
    - Performs a column comparison using the specified sort order for all Unicode columns.

- Does not perform a binary comparison.

- If no sort order is specified, rs_subcmp performs a simple binary comparison on each Unicode column of the primary and replicate row.

Schema types and schema subtypes

Table 7-4 and Table 7-5 list the schema and schema subtypes supported by rs_subcmp.

Table 7-4: Schema types supported by rs_subcmp

Type	Description
A	All aliases in the database.
D	All defaults in the database.
E	All user-defined datatypes in the database.
G	All groups in the database.
R	All rules in the database.
T	All user tables in the database. Includes table elements such as indexes, keys, constraints, and triggers.
U	All users in the database.
V	All views in the database.
P	All procedures in the database.

Table 7-5: Schema subtypes supported by rs_subcmp

Type	Description
c	Constraint
d	Bind default
f	Foreign key
g	Grant
i	Index
m	Procedure mode
p	Primary key
r	Bind rule
t	Trigger

Normalization options for faster comparison

Table 7-6 lists the normalization options for faster comparison supported by rs_subcmp.

Table 7-6: Normalization options supported by rs_subcmp

Normalization option	Description
lsb	Normalizes all byte-order-dependent data to lsb-first (little-endian) byte order.
msb	Normalizes all byte-order-dependent to msb-first (big-endian) byte order.
unicode	Normalizes the character data to Unicode (UTF-16).
unicode_lsb	Normalizes lsb in conjunction with Unicode for platform independence.
unicode_msb	Normalizes msb in conjunction with Unicode for platform independence.

CHAPTER 8 Replication Server System Tables

This chapter lists the system tables in the Replication Server System Database (RSSD) or Embedded RSSD (ERSSD). System tables are stored in a dedicated database in the system Adaptive Server or Adaptive Server Anywhere.

Access to the system tables is restricted to users with sa permission, or members of the rs_systabgroup group. The system tables are maintained by RCL commands. Avoid direct manipulation, except with rs_config. You can alter rs_config using the configure replication server command at the Replication Server.

For more information about the rs_systabgroup group, see repserver on page 500. For information about configure replication server, see configure replication server on page 166.

The system tables include the user-defined datatype rs_id that is defined as binary(8). It is used for columns that hold object names. For more information about identifiers, see "Identifiers" on page 33.

The rs_lastcommit and rs_threads system tables are documented in this chapter, although these tables are created and stored in each user database, not in the RSSD or ERSSD.

rs_articles

Description Stores information about articles known to this Replication Server.

Column	Datatype	Description
articlename	varchar(255)	Name of the article
articleid	rs_id	Unique article ID
type	char(1)	• T – table • P – procedure
primaryname	varchar(255)	Primary table or procedure name
primaryowner	varchar(30)	Primary table owner name
objid	rs_id	ID of the corresponding replication definition
pubid	rs_id	ID of the publication to which this article belongs
requestdate	datetime	Date and time the article was added to the publication
minvers	int	Minimum Replication Server version required to support this article

Indexes

- Unique clustered index on (articlename, pubid)
- Unique index on (articleid)

rs_classes

Description Stores the names of function-string classes and error classes.

Column	Datatype	Description
classname	varchar(30)	Class name
classid	rs_id	ID for this class
classtype	char(1)	One of the following values: • F – function-string class • E – error class • D – datatype class
prsid	int	ID of the site where this class is primary
parent_classid	rs_id	ID for the parent class if this is a derived class 0 if this is a base class; default is 0
attributes	int	0x01 – Default class For rs_default_function_class and rs_db2_function_class, the default is 1. Otherwise, the default is 0.

Indexes

- Unique clustered index on (classname, classtype)
- Unique index on (classid)

rs_columns

Description Contains information about the columns of replication definitions.

Column	Datatype	Description
prsid	int	Primary Replication Server for this object
objid	rs_id	Table/function replication definition ID or function ID this column belongs to
colname	varchar(255)	Column or parameter name
colnum	smallint	Column number
coltype	tinyint	Datatype of the column or parameter: • 0 – char • 1– binary • 4 – text • 5 – image • 6 – tinyint • 7 – smallint • 8 – int • 9 – real • 10 – float • 11 – bit • 12 – datetime • 13 – smalldatetime • 14 – money • 15 – smallmoney • 16 – numeric • 17 – decimal • 18 – varchar • 19 – varbinary • 25 – unichar • 27 – date • 28 – time • 29 – unitext • 30 – bigint • 31 – usmallint • 32 – uint • 33 – ubigint • 110 – univarchar
length	int	Length of the declared data

Column	Datatype	Description
searchable	tinyint	1 if searchable key, 0 if not
primary_col	tinyint	1 if primary key, 0 if not
fragmentation	tinyint	1 if fragmentation key, 0 if not
rowtype	tinyint	1 if row is to be replicated, 0 if not
status	int	Mask, can be one or more of the following: • 0x01 – column is declared an identity column • 0x04 – column is an rs_address datatype • 0x08 – column has a status of replicate_if_changed • 0x10 – column allows null values in the replicate table (only for text, unitext, or image columns) • 0x20 – column is sent to standby connection (only in internal replication definitions) • 0x40 – column is marked as dropped from the internal replication definition (only in internal replication definitions) • 0x200 – published as identity • 0x1000 – declared as Java column • 0x2000 – published as Java column
basecolnum	smallint	Column position in base replication definition. Default is *colnum* value.
repl_colname	char(255)	Column name in replicate table. Default is *colname* value.
declared_dtid	rs_id	Datatype ID. For a user-defined datatype, this is a foreign key to the table.
publ_dtid	rs_id	Published datatype as specified in the replication definition. If no published datatype is specified, publ_dtid is equal to declared_dtid.
publ_base_coltype	tinyint	The base datatype of the published datatype. If no published datatype is specified, publ_base_coltype is equal to coltype.
publ_length	int	The maximum length of the published datatype.

Indexes

- Unique index on (objid, colname)
- Unique index on (objid, colnum)
- Unique index on (objid, basecolnum)
- Clustered index on (objid)
- Partial index on (objid, searchtable)

rs_config

Description

Holds a set of default configuration parameter values that you can modify using the configure replication server command. You also can set certain parameters for specific targets using the alter connection, alter logical connection, or alter route command.

See the *Replication Server Administration Guide Volume 1* for more information about the configuration parameters in the rs_config table.

Column	Datatype	Description
optionname	varchar(30)	Name of the parameter, for example: memory_max, cm_max_connections To view a list of these parameters with their descriptions, execute a select * statement against the rs_config table.
objid	rs_id	ID of the object this option references. If set to 0, this applies to the whole system.
charvalue	varchar(255)	Character value for parameter.
status	tinyint	This column is not used.
comments	varchar(255)	Comment about the parameter.

Indexes

Unique clustered index on (optionname, objid)

rs_databases

Description Stores database names known at a Replication Server site.

Column	Datatype	Description
dsname	varchar(30)	Data server name
dbname	varchar(30)	Database name
dbid	int	Unique identifier for the database
dist_status	tinyint	Status of the connection. Can be: • 0x1 – valid • 0x2 – suspended • 0x4 – suspended by a standby-related action • 0x8 – waiting for a marker • 0x10 – will issue dbcc ('ltm', 'ignore') • 0x20 – waiting for dump marker to initialize a standby database • 0x40 – switching related duplicate detection when *ltype* is equal to ‘P’ • 0x40 – allow switching when *ltype* is equal to ‘L’ • 0x80 – temporarily not doing any grouping
src_status	tinyint	Status of the source. Can be: • 0x1 – valid • 0x2 – suspended • 0x4 – suspended by a standby-related action
attributes	tinyint	One of the following values: • 1 – distribution • 2 – source
errorclassid	rs_id	Error class for this database
funcclassid	rs_id	function-string class for this database
prsid	int	ID of Replication Server managing this database
rowtype	tinyint	Indicates the row type: • 1 – row is replicated • 0 – row not replicated
sorto_status	tinyint	Indicates if the sort order check has been completed. One of the following values: • 0 – not checked • 1 – checked
ltype	char(1)	The type of database this row represents. One of the following values: • P – physical database • L – logical database connection

Column	Datatype	Description
ptype	char(1)	The type of database in a warm standby application. One of the following values: • A – the active database • S – the standby database • L – the logical database connection
ldbid	int	The dbid for the logical connection the database is associated with. If there is no logical connection, ldbid is the same as dbid.
enable_seq	int	The sequence number used during an active database switch or the creation of a standby database.

Indexes

- Unique clustered index on (dsname, dbname, ltype)
- Unique index on (ptype, ldbid)
- Unique index on (dbid, ltype)
- Unique index on (dsname, dbname, ptype)

rs_datatype

Description

Stores attribute information for all user-defined datatypes (UDDs) in a replication definition.

Column	Datatype	Description
prsid	int	Can be: • ID of primary Replication Server • 0 for globally defined UDDs
classid	rs_id	ID of datatype class to which the datatype belongs
dtname	varchar(30)	Unique name of datatype
dtid	rs_id	Unique ID of datatype
base_coltype	tinyint	ID of base datatype for the datatype. Can be: • 0 – char • 1 – binary • 2 – longchar (not used) • 3 – longbinary (not used) • 4 – text • 5 – image • 6 – tinyint • 7 – smallint • 8 – int • 9 – real • 10 – float • 11 – bit • 12 – datetime • 13 – smalldatetime • 14 – money • 15 – smallmoney • 16 – numeric • 17 – decimal • 18 – varchar • 19 – varbinary • 21 – sensitivity • 25 – unichar • 27 – date • 28 – time • 29 – unitext

Column	**Datatype**	**Description**
		• 30 – bigint • 31 – usmallint • 32 – uint • 33 – ubigint • 101 – numeric (literal) • 102 – money (literal) • 103 – real (literal) • 104 – float (literal) • 105 – identity (literal) • 106 – timestamp (literal) • 107 – sensitivity (literal) • 110 – univarchar
length	int	Maximum length of a value of the datatype. For UDDs with masks defined as decimal or money, the value is the maximum precision plus four.
status	int	Status. (See the status column in the rs_columns table.)
length_err_act	tinyint	Action to be taken if value exceeds length identified in length. Can be: • 1 – error • 2 – continue • 3 – truncate left • 4 – truncate right • 5 – round up • 6 – round up and continue on error • 7 – round up and use default on error • 8 – round up and use minimum on error • 9 – round up and use maximum on error • 10 – round down • 11 – round down and continue on error • 12 – round down and use default on error • 13 – round down and use minimum on error • 14 – round down and use maximum on error
mask	varchar(255)	Datatype mask. Datatype must have base datatype of char for non-null mask.
scale	int	Maximum number of digits to the right of decimal point. Valid only for masks of money or decimal.

Column	Datatype	Description
default_len	tinyint	Length of value in default_val column.
default_val	binary(255)	Default value. Supplies missing components for target value during translation to this datatypes.
delim_pre_len	tinyint	Length of delim_pre value.
delim_pre	binary(30)	Postfixing character or character string used when mapping a non-Java value into a function string. An empty string if the delimiter prefix for the base datatype is used.
delim_post_len	tinyint	Length of delim_post.
delim_post	binary(30)	Postfixing character or character string used when mapping a non-Java value into a function string. An empty string if the delimiter prefix for the base datatype is used.
min_boundary_len	tinyint	Length of value in min_boundary column. • 1 – error • 2 – use default • 3 – use minimum • 4 – use maximum
min_boundary	binary(255)	Minimum acceptable value for datatype.
min_boundary_err_act	tinyint	Action to be taken if the value exceeds the minimum boundary set by min_boundary. Can be: • 1 – error • 2 – use default • 3 – use minimum • 4 – use maximum
max_boundary_len	tinyint	Length of value in max_boundary.
max_boundary	binary(255)	Maximum acceptable value for datatype.
maximum_boundary_err_act	tinyint	Action to be taken if a value exceeds the maximum boundary set by max_boundary. Can be: • 1 – error • 2 – use default • 3 – use minimum • 4 – use maximum

Column	Datatype	Description
rowtype	tinyint	Indicates whether a row is local to Replication Server or distributed to all Replication Servers in the domain. Can be: • 0 – local • 1 – global

Indexes

- Unique index on (dtid)
- Unique index on (name)
- Non-unique index on (classid)
- Non-unique index on (prsid)

rs_dbreps

Description

Stores all information about database replication definitions except name sets. It is replicated to all sites with a version number of 12.6 or later.

Column	Datatype	Description
dbrepid	rs_id	Database replication definition ID
dbrepname	varchar (255)	Database replication definition name
prsid	int	Primary Replication Server ID
dbid	int	Primary database ID
ownerid	rs_id	Replication Server user who created the database replication definition
requestdata	datetime	Time the database replication definition was created
status	int	Bitmap of subset content
minvers	int	Earliest version of Replication Server to which this table can be replicated.

Indexes

Unique indexes on (dbrepid, dbid, and dbrepname).

rs_dbsubsets

Description

Stores the name sets for database replication definitions. It is replicated to all sites with a version number of 12.6 or later.

Column	Datatype	Description
dbrepid	rs_id	Database replication definition ID
prsid	int	Primary Replication Server ID
type	char	Item type: • T – table name. • F – function name. • X – transaction name. • P – system procedure name.
owner	varchar (30)	Owner name of a table or function, or the user name that executed a transaction or system procedure. An * indicates all owners or users.
name	varchar (255)	Table, function, transaction, or system procedure name. An * indicates all tables, functions, transactions, and system procedures.

Indexes

Unique index on (dbrepid, subtype, owner, and name).

rs_diskaffinity

Description

Stores information about the affinity between disk partition and database connection or route.

Column	Datatype	Description
partition_id	int	Partition ID assigned by Replication Server
dbid_or_siteid	int	An ID for a Replication Server or database
status	int	Status of the affinity. Valid values are: • 0x01 - valid • 0x02 - obsolete

Indexes

Unique clustered index on (dbid_or_siteid)

rs_diskpartitions

Description

Stores information about the disk partitions that Replication Server uses for stable message queues.

Column	Datatype	Description
name	varchar(255)	Operating system name for the disk device
logical_name	varchar(30)	User-assigned name for the partition
id	int	Partition ID assigned by Replication Server
num_segs	int	Total size of the partition in segments
status	int	Status of the disk partition. Valid values are: • 1 – online • 2 – partition is being dropped
vstart	int	Offset at which Replication Server starts writing to the partition (in MB)

Indexes

- Unique clustered index on (logical_name)
- Unique index on (name)

rs_erroractions

Description

Maps a data server error number to an action to be taken by a Replication Server.

Column	Datatype	Description
ds_errorid	int	Data server error number
errorclassid	rs_id	Error class ID (see rs_classes)
action	tinyint	Action to take when error occurs: • 1 – ignore the error • 2 – stop replication • 3 – output a warning message • 4 – write an entry in the exceptions log • 5 – retry the transaction and then log the transaction if it still fails • 6 – retry the transaction a certain number of times and then stop replication if it still fails
prsid	int	Site where this row is primary

Indexes

- Unique index on (ds_errorid, errorclassid)
- Clustered index on (errorclassid)

rs_exceptscmd

Description

Stores the information used to retrieve the text of transactions from the exceptions log. The text, stored in the rs_systext system table, includes:

- Source command – the text of the user transaction received by Replication Server.
- Output command – the text of the transaction that Replication Server prepared for the database from function strings. The output command can be either a language command or an RPC.

rs_exceptscmd has one row for each source command or output command.

Column	Datatype	Description
sys_trans_id	rs_id	System-assigned transaction ID for the transaction
src_cmd_line	int	Command-line number of the source within the logged transaction
output_cmd_index	int	Line number of the output command within the logged transaction
cmd_type	char(1)	Command type: • S – source command • L – language output command • R – RPC output command
cmd_id	rs_id	Index into rs_systext

Indexes

Unique index on (cmd_id)

rs_exceptshdr

Description

Stores information about failed transactions. The source and output commands of the transactions are stored in the system tables rs_exceptscmd and rs_systext. All rows for a transaction in rs_exceptscmd and rs_exceptshdr are identified by the column sys_trans_id.

Column	Datatype	Description
sys_trans_id	rs_id	System-assigned transaction ID for this transaction
rs_trans_id	binary(120)	Replication Server-generated unique transaction ID
app_trans_name	varchar(30)	User-specified transaction name
orig_siteid	int	ID of the origin database
orig_site	varchar(30)	Data server name for the origin database
orig_db	varchar(30)	Name of the origin database
orig_time	datetime	Time the transaction was initiated
orig_user	varchar(30)	User who submitted the transaction at the origin site
error_siteid	int	ID of the site where the error occurred
error_site	varchar(30)	Name of the data server where the error occurred
error_db	varchar(30)	Name of the database where the error occurred
log_time	datetime	Time the error occurred
ds_error	int	Data server error number
ds_errmsg	varchar(255)	Data server error message
error_src_line	int	Line number of the command that caused the error
error_proc	varchar(255)	Procedure during which the error occurred
err_output_line	int	Line number of the output command that caused the error
log_reason	char(1)	Why the transaction was logged: • O – indicates an orphan transaction in the DSI queue • E – a data server error mapped to LOG or RETRY_LOG • S – indicates the transaction was skipped because the resume connection command was executed with the skip transaction option • D – the transaction was logged by a sysadmin log_first_tran command
trans_status	smallint	Transaction status—one or more of the following: • 0x0001 – orphan transaction • 0x0002 – logged transaction was going to primary site • 0x0004 – conflicting transaction
retry_status	smallint	Retry status for the transaction—one of the following: • 1 – retry succeeded • 2 – transaction has not committed
app_usr	varchar(30)	Name of the user who applied the transaction at the error site

Column	Datatype	Description
app_pwd	varchar(30)	Password of the user who applied the transaction at the error site

Indexes

Unique index on (sys_trans_id)

rs_exceptslast

Description

Stores the origin ID, secondary queue ID, and associated information about the last logged transaction written into the exceptions log.

Column	Datatype	Description
error_db	int	Database where the error occurred
origin	int	Origin database of the transactions
origin_qid	binary(36)	qid of the last transaction from this origin
secondary_qid	binary(36)	Secondary qid of the last logged transaction from this origin
status	tinyint	Status of the transaction: • 0 – Valid: no transactions were lost for this origin • 1 – Detecting losses: you should determine if any transactions have been lost in this origin • 2 – Rejecting messages after loss detected: transactions were probably lost for this origin
origin_time	datetime	Time at origin for the transaction
log_time	datetime	Time the transaction was logged
lorigin	int	Logical database where the message originated

Indexes

- Unique index on (error_db, origin)
- Unique index on (error_db, origin, status)

rs_funcstrings

Description

Stores the function strings associated with each function.

Column	Datatype	Description
prsid	int	Site where this row is primary
classid	rs_id	Class the function string belongs to
funcid	rs_id	Function this string is for
name	varchar(255)	Function string name
fstringid	rs_id	ID for this function string
attributes	smallin	Attributes of the function string: • 0x01 – conflicting function • 0x02 – RPC • 0x04 - altered • 0x10 – default input • 0x20 – default output • 0x40 – writetext output is used for an rs_writetext function string • 0x80 – writetext output is used with the with log option for an rs_writetext function string • 0x100 – a function string for an rs_writetext, rs_textptr_init, or rs_get_textptr function • 0x200 – writetext output is used with the no log option for an rs_writetext function string • 0x400 – function string includes one or more variables that will access the values of non-key columns • 0x800 – the *rs_default_fs* system variable was used in output language template • 0x1000 – none output is used for an rs_writetext function string
parameters	smallint	Number of parameters in this function string
param_hash	int	Hash value of input template
expiredate	datetime	Date the function string should expire. This is used for dynamic function string expiration
rowtype	tinyint	1 if this row is replicated, 0 if not
minvers	int	Minimum version required to support the function string. This means that if a function string has minvers value of 15.0, it will not replicate to sites below 15.0

Indexes

- Unique clustered index on (classid, funcid, name)
- Unique index on (fstringid)

rs_functions

Description Stores information about Replication Server functions.

Column	Datatype	Description
prsid	int	Site where the function is primary
funcname	varchar(255)	Name of the function
funcid	rs_id	ID of the function
objid	rs_id	Object to which the function applies. NULL_OBJECT_ID (0x00000000) is stored in this column for class-scope functions.
conflicting	tinyint	1 if the function is conflicting, 0 if not
userdefined	bit	1 if this is a user-defined function, 0 if not
rowtype	tinyint	1 if this row is replicated, 0 if not

Indexes

- Clustered index on (objid)
- Unique index on (objid, funcname)
- Unique index on (funcid)

rs_idnames

Description

Stores the names of Replication Servers and databases known to the ID server. This table is relevant only at the ID Server site.

Column	Datatype	Description
name1	varchar(30)	Replication Server or data server name
name2	varchar(30)	Database name; “” for a Replication Server
type	int	Replication Server or database: • 8 – Replication Server • 9 – database
id	int	Unique ID assigned to the Replication Server or database
ltype	char(1)	The type of the database: • P – Physical database • L – Logical database

Indexes

Unique clustered index on (name1, name2, ltype)

rs_ids

Description — Stores the last ID used for various types of objects.

Column	Datatype	Description
typename	varchar(30)	Name of this object type. For example, “subscriptions,” “objects”
objid	int	Last ID used for this object type
objtype	tinyint	• Object type: • 1 – Subscriptions • 2 – Objects • 3 – Classes • 4 – Users • 5 – Functions • 6 – Function strings • 7 – Error log • Exception log types: • 12 – Reject transaction • Site ID types: • 8 – Replication Server ID • 9 – Database ID • Stable queue parameters: • 10 – Disk partition IDs • Counter used by subscriptions module: • 13 – Counter for subscriptions module • Recovery manager IDs: • 14 – Recovery ID type • 15 – Rematerialization ID • 16 – Publication ID • 17 – Article ID • 18 – where clause ID • 19 – UDD ID

Indexes — Unique clustered index on (objtype)

rs_lastcommit

Description

Replication Server uses the information in this table to find the last transaction committed from each data source.

The rs_lastcommit table is stored in each user database, not in the RSSD.

Column	Datatype	Description
origin	int	ID number for the primary database a row represents.
origin_qid	binary	Identifies the last committed transaction in the stable queue for the origin database.
secondary_qid	binary	If a subscription materialization queue exists for the origin database, this column contains the last transaction in that queue that has been committed in the replicate database.
origin_time	datetime	Time at origin for the transaction.
dest_commit_time	datetime	Time the transaction was committed at the destination.
pad1	binary(255)	Filler to pad the row so only one row fits on a database page.
pad2	binary(255)	Filler to pad the row so only one row fits on a database page.
pad3	binary(255)	Filler to pad the row so only one row fits on a database page.
pad4	binary(255)	Filler to pad the row so only one row fits on a database page.
pad5	binary(255)	Filler to pad the row so only one row fits on a database page.
pad6	binary(255)	Filler to pad the row so only one row fits on a database page.
pad7	binary(255)	Filler to pad the row so only one row fits on a database page.
pad8	binary(83)	Filler to pad the row so only one row fits on a database page.

Indexes

Unique clustered index on (origin)

rs_locater

Description: Stores the last locator field received by stable queues from each of their senders.

Column	Datatype	Description
sender	int	Sender site ID
type	char(1)	Who is using this row: • R – RSI (route) • D – distributor locater used for subscriptions • E – executor for Replication Agent • U – locator at last system upgrade • W – distributor locator used for a warm standby application
locater	binary(36)	Last queue ID received from this sender

Indexes: Unique clustered index on (sender, type)

rs_maintusers

Description Stores the user login names and passwords Replication Server uses to access other Replication Servers and data servers.

Column	Datatype	Description
destid	int	Site ID for the Replication Server or database to be logged into
username	varchar(30)	User name for the Replication Server RSI user or for the database maintenance user
password	varchar(30)	Password
use_enc_password	int	• 0 – use normal passwords • 1 – use encrypted passwords
enc_password	varchar(66)	Encrypted user password

Indexes Unique clustered index on (destid)

rs_msgs

Description

Stores the localized error messages used during installation and by some Replication Server stored procedures.

Column	Datatype	Description
msgnum	int	Unique ID number for the message
langname	char(30)	Local language name of this version of the message text. Corresponds to the *@@language* global variable in the RSSD Adaptive Server.
msgtxt	varchar(255)	Text of the message, in the localized language.

Indexes

Unique clustered index on (msgnum, langname)

rs_objects

Description Stores replication definitions, one per row.

Column	Datatype	Description
prsid	int	Primary Replication Server where this object was created
objname	varchar(255)	Object name
objid	rs_id	Object ID
dbid	int	Unique ID for data server and database
objtype	char(1)	One of the following object types: • R – table replication definition • F – function replication definition
attributes	int	Mask, can be one or more of the following: • 0x01 – Generate dynamic function strings • 0x02 – Replication definition has fragments • 0x04 – Minimum columns enabled for replication definition • 0x08 – Replication definition has identity column • 0x10 – replicate_if_changed status • 0x20 – Replication definition has a drop pending • 0x40 – Replication definition has text, unitext, or image column • 0x80 – Replication definition is used by a standby • 0x0100 – Replication definition's columns are sent to standby database • 0x0200 – Replication definition is propagated to Replication Servers Version 11.0.x or earlier • 0x0400 – Replication definition has been used as a base replication definition for the primary table • 0x0800 – Replication definition is internal only • 0x1000 – Object or column names differ in the primary and replicate tables • 0x4000 – replication definition has column-level translations • 0x8000 – replication definition has columns declared with UDDs
ownertype	char(1)	Type of owner of this object: • U – user • S – System
crdate	datetime	Date and time created
parentid	rs_id	Reserved for future use.
ownerid	rs_id	ID of the user who created this object
rowtype	tinyint	1 if row is replicated, 0 if not
phys_tablename	varchar(255)	Primary table name – used when communicating with data server about this object

Column	Datatype	Description
deliver_as_name	varchar(255)	Name of the replicate table or stored procedure
phys_objowner	char(30)	Name of the primary table owner, as specified in replication definition. Blank if the table owner is not specified.
repl_obj_owner	char(30)	Name of the replicate table owner, as specified in replication definition. Blank if the table owner is not specified.
has_baserepdef	rs_id	If this is not a base replication definition, the value of has_baserepdef matches that of objid for the base replication definition. Or, has the following value: 0x00 - Base replication definition
minvers	int	Specifies the minimum version of a replication definition, and thus the Replication Server to which it can propagate. Can be: • 1200 – propagates to Replication Server version 12 or later • 1150 – propagates to Replication Server version 11.5 or later • 1000 or 0 (zero) – propagates to any Replication Server • 0 (zero) – for function and system replication definitions

Indexes

- Unique clustered index on (objname)
- Unique index on (dbid, phys_tablename, phys_objowner, objtype, has_baserepdef)
- Unique index on (objid)
- Unique index on (objid, dbid)

rs_oqid

Description   Stores the last queue ID received from an origin site.

Column	Datatype	Description
origin_site_id	int	Site ID of the origin site
q_number	int	Queue number
q_type	int	Queue type
origin_q_id	binary(36)	Command ID at the origin database
local_q_id	binary(36)	Local ID for the queue
valid	int	Validation status: • 0 – valid • 1 – detecting losses • 2 – rejecting messages after loss detected
origin_lsite_id	int	Site ID of the logical database of the origin site

Indexes

- Unique clustered index on (origin_site_id, q_number, q_type)

rs_publications

Description

Stores information about publications known to this Replication Server.

Column	Datatype	Description
prsid	int	Primary Replication Server where the publication was created
pubname	varchar(255)	Name of the publication
pubid	rs_id	Unique publication ID
pdbid	int	Unique ID for the publication’s primary data server and database
requestdate	datetime	Date and time the last article was added to the publication
ownerid	rs_id	ID of the user who created the publication
status	int	Publication status: • 0x00 – Invalid • 0x01 – Valid
minvers	int	Minimum Replication Server version required to support this publication

Indexes

- Unique clustered index on (pubname, pdbid)
- Unique index on (pubid)

rs_queuemsg

Description

When you dump Replication Server queues into the RSSD (using sysadmin dump_queue), the queue entries are stored in rs_queuemsg. If this table already has rows for a segment, those rows are deleted from the table before the latest rows from that segment are dumped.

Column	Datatype	Description
q_number	int	Queue number
q_type	int	Queue type
q_seg	int	Queue segment
q_blk	int	Queue block
q_row	int	Queue row
len	int	Length of the queue entry
origin_site_id	int	Origin site ID
origin_q_id	binary(36)	Queue ID assigned by the origin
origin_time	datetime	Time transaction was initiated
origin_user	varchar(30)	User who submitted transaction at origin site
tran_name	varchar(30)	Transaction name
local_q_id	binary(36)	Queue ID assigned by the local Replication Server
status	int	Message status
reserved	int	Reserved for future use
tran_len	smallint	Length of tran_id
txt_len	smallint	Length of command
tran_id	binary(120)	Transaction ID
lorigin_site_id	int	Site ID of the logical connection that is the source of the queue entries.
version	int	Release version of the message

Indexes

Unique clustered index on (q_number, q_type, q_seg, q_blk, q_row)

rs_queuemsgtxt

Description

Stores the command or text portion of messages in stable queues. Each stable queue entry is represented by one or more rows in this table. Multiple rows are needed when the length of data in the stable queue entry exceeds the maximum command field length of 255 bytes.

Column	Datatype	Description
q_number	int	Queue number
q_type	int	Queue type
q_seg	int	Segment that contains the message
q_blk	int	Block within the segment that contains the message
q_row	int	Row within the block that contains the message
q_seq	int	Sequence number of the row for this entry
txt	varchar(255)	Text of the entry
txtbin	binary(255)	Text in binary

Indexes

Unique default index on (q_number, q_type, q_seq, q_seg, q_blk, q_row)

rs_queues

Description

Stores information to allow site recovery. Used by the Replication Server stable queue manager and guaranteed delivery system.

Column	Datatype	Description
number	int	Queue ID. This column displays a number representing either: • The source database for an inbound queue, or • The destination database or Replication Server for an outbound queue Values correspond to entries for databases in the dbid column in the rs_databases system table and to entries for Replication Servers in the id column in the rs_sites system table.
type	int	Queue type: • 0 – outbound queue • 1 – inbound queue • large negative number – a subscription materialization queue
state	int	Current state of this queue: • 0 – failure • 1 – active • 2 – deleting
twosave	int	Indicates the number of seconds the Replication Server maintains an SQM segment after all messages in the segment have been acknowledged by targets. A setting of -1 indicates a strict setting.
truncs	int	The number of truncation points

Indexes

Unique clustered index on (number, type)

rs_recovery

Description

Logs actions that must be performed by Replication Server upon recovery, if there is a failure.

Column	Datatype	Description
action	int	Represents the recoverable actions: • 1 – create_route • 2 – drop_route • 3 – standalone mode • 4 – rebuild queues • 5 – log recovery • 6 – restart LTM at the top of the log • 7 – create standby • 8 – switch active • 9 – strict save interval for DSI or materialization queue • 10 – quit DSI secondary duplicate detection after switch active • 11 – drop standby • 12 – alter distributor locater • 13 – delete segments with replication definitions • 14 – drop pending replication definitions • 15 – drop pending table or function replication definition with reference counter • 16 – create schema replication definition (for auto-generating schema replication definition)
id	rs_id	Each row is assigned a unique ID.
seqnum	int	For actions with multiple rows, this column stores the sequence number of each row.
state	int	Contains the current state for recoverable actions that move through a finite number of states.
text	binary(255)	Data required to complete the action.
textlen	int	Length of the text data.

Indexes

Unique index on (id)

rs_repdbs

Description

Contains information about all of the databases known by a primary Replication Server. This information is stored when a subscription is entered for a database at a replicate site.

Column	Datatype	Description
dbid	int	Unique database ID
dsname	varchar(30)	Data server name
dbname	varchar(30)	Database name
controllerid	int	Managing Replication Server for this database

Indexes

- Clustered index on (controllerid)
- Unique index on (dbid)
- Unique index on (dsname, dbname)

rs_repobjs

Description

Stores autocorrection flags for replication definitions at replicate Replication Servers. Set the flag to on or off using the set autocorrection command.

Column	Datatype	Description
objid	rs_id	Replication definition object ID
dbid	int	ID of the database where the replicate data is stored
attributes	int	Valid value: 0x01 – autocorrection flag is on

Indexes

Unique clustered index on (objid, dbid)

rs_routes

Description Stores routing information about network traffic.

Column	Datatype	Description
dest_rsid	int	ID of a data server or Replication Server
through_rsid	int	Destination is reached through this Replication Server. For a direct route, the value of through_rsid is the same as that of dest_id.
source_rsid	int	Replication Server where this route is defined
status	tinyint	Status of the route: • 1 – being initialized • 2 – route is valid at this site (route is valid when status is 2 at both the source and destination Replication Servers) • 3 – dropping this route gracefully • 4 – dropping this route immediately
suspended	tinyint	One of the following values: • 0 – route is active • 1 – route is suspended • 2 – route is being rebuilt. In the process of setting the truncation point. • 3 – route is suspended. In the process of setting the truncation point. • 8 (mask) – for an RSI outbound queue, instructs the replicate Replication Server to set the locater field in the rs_locater table to 0, for this sending Replication Server.
src_version	int	Version of source Replication Server for this route. Note that this version is the RSI version (not what appears in the rs_config stored procedure under current_rssd_version). • 1000 – version assigned to any pre-10.1 Replication Server • 1010 – version 10.1 • 1100 – version 11.0 • 1150 – version 11.5 • 1200 – version 12.0 Refer to the *Release Bulletin for Replication Server* for any additional supported version numbers.

Indexes Unique clustered index on (dest_rsid, source_rsid)

rs_routeversions

Description Stores version information about the Replication Servers on each end of a route.

Column	Datatype	Description
dest_rsid	int	ID of the destination Replication Server
source_rsid	int	ID of the source Replication Server where this route is defined
dest_rssd_id	int	ID of the RSSD of the destination Replication Server
route_version	int	The minimum site version of the destination and source Replication Server
min_path_version	int	Reserved for future use
marker_serial_no	int	For internal use
status	int	Route status: • 0x00 – Valid • 0x01 – Route upgrade/recovery in progress, or route upgrade/recovery needed. • 0x02 – Route upgrade/recovery complete. This is a temporary status used by Replication Manager.
proposed_version	int	New route value in transition

Indexes Unique clustered index on (dest_rsid, source_rsid)

rs_rules

Description

Stores subscription rules. This table has one row for each term in a subscription clause.

Column	Datatype	Description
prsid	int	Primary Replication Server for this object
subid	rs_id	ID of the subscription this rule applies to. Or, for a subscription to an article, the ID of the where clause to which this rule applies.
objid	rs_id	ID for the table or function replication definition for this subscription
dbid	int	ID for the database where the subscribed data is stored
subtype	int	Subscription type: • 0x01 – Range subscription • 0x02 – Equality subscription • 0x80 – Article subscription
primary_sre	int	If set, the subscription should be included in the subscription resolution engine at the primary Replication Server
replicate_sre	int	If set, the subscription should be included in the subscription resolution engine at the replicate Replication Server
colnum	smallint	The value of the base column number
valuetype	tinyint	Datatype of operand, for example, SYBCHAR
low_flag	tinyint	Bitmap for the type of the low value: • 0x01 – exclusive • 0x02 – inclusive • 0x04 – infinity • 0x08 – equality • 0x20 – rs_address
high_flag	tinyint	Bitmap for the type of the high value: • 0x01 – exclusive • 0x02 – inclusive • 0x04 – infinity • 0x08 – equality • 0x20 – rs_address
low_len	int	Length of low value
high_len	int	Length of high value
low_value	binary(255)	Binary representation of low value
high_value	binary(255)	Binary representation of high value
dtid	rs_id	ID of the declared datatype of the columns as defined in the replication definition.

Indexes	• Unique index on (subid, colnum, primary_sre, replicate_sre, subtype) • Unique index on (subid, colnum) • Clustered index on (objid, subtype, dbid)

rs_segments

Description

Replication Server uses raw disk space to store message data. This table holds information about the allocation of each segment.

Column	Datatype	Description
partition_id	int	Unique ID for the partition
q_number	int	Queue that this partition belongs to
q_type	int	Type of this queue
partition_offset	int	Offset of segment within partition
logical_seg	int	Offset of segment within queue
used_flag	int	Current status of segment: • 0 – inactive • 1 – active • *n* – save interval: *n* indicates the actual time (measured in seconds from a base date) when this segment can be deleted
version	int	Current version of the segment. The version number increases after each use.
flags	int	Set to 1 on the last segment of the DSI queue after switch active

Indexes

Unique clustered index on (partition_id, partition_offset)

rs_sites

Description Stores the names of Replication Servers known at a site.

Column	Datatype	Description
name	varchar(30)	Replication Server name
id	int	Site ID assigned to this Replication Server
status	tinyint	Not used

Indexes
- Unique index on (name)
- Unique clustered index on (id)

rs_statcounters

Description Stores descriptive information about each counter. These values do not change.

Column	Datatype	Description
counter_id	int	Unique counter identification number
counter_name	varchar(60)	Descriptive counter name
module_name	varchar(30)	Name of module to which the counter belongs
display_name	varchar(30)	Counter name used for RCL commands
counter_status	int	Counter status. Bit-mask values are: • 0x001 – internal use, does not display • 0x002 – internal use, does not display • 0x004 – sysmon (counter flushed as output of admin statistics, sysmon) • 0x008 – must sample (counter sampled at all times) • 0x010 – no reset (counter is never reset) • 0x020 – duration (counter records amount of time to complete an action, usually in .01 seconds) • 0x040 – internal use, does not display • 0x080 – keep old (previous value of counter retained, usually to aid calculation during next observation period) • 0x100 – internal use, does not display • 0x200 – observer • 0x400 – monitor • 0x800 – internal use, does not display
description	varchar(255)	Description of counter

Indexes Unique, clustered key rs_key_statcounters on (counter_id)

rs_statdetail

Description Stores counter metrics that have been flushed to the RSSD.

Column	Datatype	Description
run_id	rs_id	Number assigned to the run or observation period
instance_id	int	An ID that identifies a module instance. Counters are grouped by modules. A module may have one instance or multiple instances. Defined module IDs are used when available. For example, the instance_id for a DSI module is the database ID associated with the DSI.
instance_val	int	An ID that identifies a module instance when instance_id can not identify it uniquely.
counter_id	int	Unique counter identification number
counter_obs	int	Number of observations
counter_total	int	Total of observed values for the run or observation period
counter_last	int	Last observed value for the run or observation period
counter_max	int	Maximum observed value for the run or observation period
label	varchar(255)	Descriptive information about the module instance associated with the counter, such as the data server and database name.

Indexes Unique, nonclustered key rs_key_statdetail on (run_id, instance_id, instance_val, counter_id)

rs_statrun

Description Stores descriptive information about each observation period or run.

Column	Datatype	Description
run_id	rs_id	Number assigned to an observation period or run
run_date	datetime	Date and time of observation period or run
run_interval	int	Duration of observation period or run in seconds
run_user	varchar(30)	Name of user who flushed the counters to the RSSD
run_status	int	Status of run

Indexes	Unique, nonclustered key rs_key_statdetail on (run_id)

rs_subscriptions

Description Stores information about subscriptions, triggers, and fragments.

Column	Datatype	Description
subname	varchar(30)	Name of the subscription, trigger, or fragment.
subid	rs_id	ID for this subscription or fragment.
type	int	Object type: • 0x00 – Subscription • 0x01 – Range subscription • 0x02 – Equality subscription • 0x04 – Entire table • 0x08 – Subscription for publication • 0x40 – Database subscription • 0x80 – Subscription for article
objid	rs_id	ID for the table replication definition, function replication definition, article, or publication for this subscription. Or, ID for fragment, or event for this trigger.
dbid	int	ID of the database this object belongs to.
pdbid	int	For system table replication and publication or article subscriptions, the value of pdbid is the ID of the primary database for the replication definition. Otherwise, value is 0.
requestdate	datetime	Date and time the last DDL request (create, drop, alter) was entered.
pownerid	rs_id	User ID at the primary Replication Server.
rownerid	rs_id	User ID at the replicate Replication Server.

Column	Datatype	Description
status	int	• Byte 1 holds the replicate database materialization status: • 0x01 – Subscription is new • 0x02 – Bulk subscription is activating or atomic/non-atomic subscription has completed building materialization queue • 0x04 – Bulk/non-atomic subscription is active • 0x08 – Bulk subscription is validating or non-atomic has materialized • 0x10 – Subscription is valid • 0x40 – Subscription is valid at the standby • 0x40 – Subscription removed at standby • 0x80000000 – Database subscription is using dump marker to coordinate materialization • Byte 2 holds the primary database dematerialization status: • 0x100 – New • 0x0200 – Activating • 0x0400 – Active • 0x0800 – Valid • Byte 3 holds the replicate database dematerialization status: • 0x00010000 – Dematerializing at replicate • 0x00020000 – Removing at replicate • 0x00100000 – Dematerializing at primary • Byte 4 holds suspect or rematerialization status for a publication subscription: • 0x02000000 – Suspect because of switch active • 0x04000000 – Suspect on drop at standby • 0x10000000 – The article subscriptions within this publication subscription are materializing one at a time • 0x20000000 – In the process of creating new article subscriptions • 0x40000000 – include truncate table
recovering	int	Subscription recovery status: • 0x0 – Subscription is OK • 0x1 – Recovering • 0x2 – Pending
error_flag	int	If set, subscription is unrecoverable
materializing	int	If set, subscription is materializing
dematerializing	int	If set, subscription is dematerializing

Column	Datatype	Description
primary_sre	int	If set, the subscription should be included in the subscription resolution engine at the primary Replication Server
replicate_sre	int	If set, the subscription should be included in the subscription resolution engine at the replicate Replication Server
materialization_try	int	Number of times this atomic materialization has been tried
method	int	Method for materializing the subscription: • 0x00 – Default method • 0x01 – Atomic • 0x02 – Bulk • 0x04 – Suspend • 0x08 – Incremental • 0x10 – Non-atomic • 0x80 – Bulk materialization with suspended standby DSI **Note** For function replication definitions, this column is always set to 0x02 (bulk)
generation	binary	Generation number for the origin queue ID of the materialization queue
parentid	rs_id	ID for the subscription for a publication if the current subscription is for an article.
security	int	Security settings: • 0x001 – unified_login is “required” • 0x002 – mutual_auth is “required” • 0x004 – msg_confidentiality is “required” • 0x08 – msg_integrity is “required” • 0x10 – msg_origin_check is “required” • 0x20 – msg_reply_detection is “required” • 0x40 – msg_sequence_check is “required” Default: 0
mechanism	char(30)	Name of security mechanism Default: NULL

Indexes

- Unique clustered index on (subid)
- Unique index on (objid, dbid, subname)
- Unique index on (subid, recovering, error_flag, materializing, dematerializing, primary_sre, replicate_sre)
- Unique index on (subid, status)

rs_systext

Description

Stores the text of repeating groups for various other tables such as rs_funcstrings.

Column	Datatype	Description
prsid	int	Replication Server where the object is defined
parentid	rs_id	ID of the object this text is for
texttype	char(1)	Type of object this row is for: • S – input template for function string • O – output template for function string • C – command from a logged transaction in the exceptions log
sequenc e	int	Sequence of the text
textval	varchar(255)	The text

Indexes

Unique clustered index on (parentid, texttype, sequence)

rs_threads

Description

Replication Server uses the information in this table 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.

The rs_threads table is stored in each user database, not in the RSSD.

Column	Datatype	Description
id	int	The entry ID number. There are two entries for each parallel DSI thread.
seq	int	The sequence number of the last update made to this entry. The sequence number starts at 0 each time the connection is restarted.
pad1	char(255)	Filler to pad the row so that only one row fits on a database page.
pad2	char(255)	Filler to pad the row so that only one row fits on a database page.
pad3	char(255)	Filler to pad the row so that only one row fits on a database page.
pad4	char(255)	Filler to pad the row so that only one row fits on a database page.

Indexes

Unique clustered index on (id)

rs_translation

Description Stores information about class-level datatype translations.

Column	Datatype	Description
prsid	int	ID of the primary Replication Server
classid	rs_id	Function-string class ID of connection
type	char(1)	Type of translation. Can be: • D – class-level
source_dtid	rs_id	ID of source datatype
target_dtid	rs_id	ID of target datatype
target_length	int	Maximum length for a value of the target datatype
target_status	int	See status column in rs_columns table
rowtype	tinyint	Indicates whether a row is local to the Replication Server or distributed to all Replication Servers in the domain. Can be: • 0 – local • 1 – global

Indexes

- Unique, compound index on (classid, source_dtid)
- Non-unique index on (classid, prsid)

rs_users

Description

Stores a row for each user with access to the Replication Server.

Column	Datatype	Description
username	varchar(30)	Name of the user
uid	rs_id	ID of the user
password	varchar(30)	Password
permissions	smallint	Mask indicating roles a user can have: • 0x0001 – sa • 0x0002 – connect source • 0x0004 – create object • 0x0008 – primary subscribe
use_enc_password	int	• 0 – use normal passwords • 1 – use encrypted passwords
enc_password	varchar(66)	Encrypted password

Indexes

- Unique index on (username)
- Unique index on (uid)

rs_version

Description

Stores version number information for the replication system. At local Replication Servers, only the local version number and the system-wide version number are stored. At the ID Server, version information is stored for all Replication Servers in the replication system.

Column	Datatype	Description
siteid	int	ID number of the Replication Server: • 0 – site ID for the system-wide version number • 1 – site ID for the site version number • *n* – site ID of individual Replication Servers
version	int	Version number: • 1000 – version 10.0 (assigned to any Replication Server whose version is unknown) • 1003 – version 10.0.3 • 1011 – version 10.1.1 • 1100 – version 11.0 • 1101 – version 11.0.1 • 1102 – version 11.0.2 • 1103 – version 11.0.3 • 1150 – version 11.5 • 1200 – version 12.0 • 1210 – version 12.1 • 1250 – version 12.5 • 1260 – version 12.6 • 1500 – versions 15.0, 15.0.1 See the *Release Bulletin for Replication Server* for any additional supported version numbers.

For more information about system-wide version numbers, see admin security_property on page 67.

Indexes

Unique clustered index on (siteid)

rs_whereclauses

Description

Stores information about where clauses used in articles known to this Replication Server.

Column	Datatype	Description
articleid	rs_id	ID of the article included in this where clause
wclauseid	rs_id	ID of this where clause
type	int	• 0x01 – Range • 0x02 – Equality

Indexes

Unique clustered index on (wclauseid)

CHAPTER 9 Replication Monitoring Services API

This chapter contains the reference pages for the Replication Monitor Service (RMS) API. Table 9-1 provides a brief description of the API commands.

Table 9-1: RMS API commands

Command	Description
add event trigger	Sets up a trigger, such as a process or a script, that is executed by the RMS when a specific event occurs.
add server	Adds a server to be monitored by the RMS.
configure component	Returns configuration parameters for a component; or sets the value of the specified configuration parameter. Components are monitored objects within a server, including Replication Server and Adaptive Server Enterprise.
configure RMS	Returns the configuration parameter information for the RMS, or sets the value of a specified RMS configuration parameter.
configure server	Returns configuration parameter information for a Replication Server or Replication Agent, or sets the value of a specified configuration parameter. Also retrieves and sets RMS-specific parameters.
connect to server	Provides a pass-through mode that enables you to send commands to a server that is monitored by the RMS. Result sets generated by commands are passed back to the client.
create group	Enables you to define a set of servers and issue commands to all members of the group.
delete group	Deletes a logical group that was added using the create group command.
disconnect server	Disconnects from a server where a pass-through connection was established.
drop event trigger	Removes a trigger that the RMS is monitoring, using the add *event* triggers command.
drop server	Drops a server that is being monitored by the RMS.
filter connection	Returns current filter settings or sets the filter setting for a connection. This command can filter either the Replication Agent thread or the DSI thread status.
get component	Returns a list of Replication Server or Adaptive Server Enterprise components that are monitored by the RMS. Components are monitored objects within a server.
get group	Returns a result set that contains either a list of the groups and a roll-up status for each group, or status of each server and a roll-up status for the specified group. Roll-up status shows the lowest status reported for a component in the groups.
get heartbeat	Retrieves a list of the heartbeat processes that have been defined in the RMS.

Command	Description
get heartbeat tickets	Retrieves a set of tickets from the rms_ticket_history table, for the heartbeat process and date and time range specified.
get network spec	Retrieves the connection information for all servers known to the RMS. This list is retrieved from the RMS's *interfaces* file or LDAP server. The list consists of the server name, host computer name, and the port number used by the server.
get rmiaddress	Retrieves the address of the Remote Method Invocation (RMI) service.
get servers	Returns a list of servers that are monitored by the RMS, and the status of the RMS environment. The RMS status is a roll-up of the monitored servers.
get status descriptions	Retrieves the list of status descriptions for a server or component.
get threads	Displays information about threads running in the Replication Server.
get triggers	Displays information about the triggers that are monitored by the RMS.
get version	Retrieves the version number of RMS.
resume component	Resumes a component in a specified server. The command resumes a DSI thread, Replication Agent thread, RepAgent thread, queue, or a route in a Replication Server.
resume Replication Agent	Resumes replication in a Replication Agent.
shutdown server	Issues a shutdown command to a server or to the RMS.
suspend component	Suspends a component in a specified server. The command suspends a DSI thread, Replication Agent thread, RepAgent thread, or route in a Replication Server.
start heartbeat	Sets up and starts a heartbeat process from a specified primary connection to a specified replicate connection.
stop heartbeat	Stops the heartbeat process between the primary and replicate databases. Optionally, truncates the rms_ticket_history table.
suspend Replication Agent	Suspends replication in a Replication Agent.
trace	Displays trace information in the RMS log file.

To use the RMS API commands, these permissions must be set for each server that is monitored by RMS:

Server	Permission
Adaptive Server	The user must have “sa” or “dbo” permissions or Replication role for any primary database.The user must have “sa” or “dbo” permissions for any RSSD database.
Replication Server	The user must have “sa” permissions.
Replication Agent	The server does not have different user permissions.
Mirror Replication Agent	The server does not have different user permissions.
DirectConnect	The user must have permission to successfully log into the back end server. The RMS does not attempt to read or write to the back end database.
ASA	The user must have permission to log into the ASA. The RMS does not attempt to read or write to the database.

Server	Permission
IQ	The user must have permission to log into the IQ server. The RMS does not attempt to read or write to the database.
Remote RMS	The server does not have different user permissions.
Open Server	The user must have permission to establish a connection to the Open Server.

add *event* trigger

Description

Adds a trigger that is executed by the RMS when a specific event occurs in the replication domain. A trigger identifies a process or script that is executed by the RMS.

Syntax

```
add {status | latency | size} trigger
     [{connection | logical connection | route | queue | rep agent |
       partition} [component_name]]
     [with primary primary_connection]
     for server_name
     {status changes to state |
       size {exceeds | falls below} size_threshold |
       latency {exceeds | falls below} latency_threshold}
     [wait wait_interval]
     [continuous continuous_flag]
     execute command
```

Parameters

status, latency, size

Type of trigger.

connection, logical connection, route, queue, rep agent, partition

Specifies the type of component to be monitored. Components are monitored objects within a server. Replication Server components are connections, logical connections, routes, queues, and partitions; Adaptive Server Enterprise components are RepAgent threads.

component_name

Specifies the name of the component to be monitored.

with primary *primary_connection*

Identifies the primary connection for a connection latency trigger. The trigger executes the script if the latency threshold between the primary connection and the replicate connection is not satisfied.

for *server_name*

Specifies the name of the server to be monitored. If the command is to add a trigger for a component, then the server is the owner of the component.

size exceeds, falls below *size_threshold*
Indicates whether the trigger should execute when the size exceeds the threshold or when it falls below the threshold.

latency exceeds, falls below *latency_threshold*
Indicates whether the trigger should execute when the latency exceeds the threshold or when it falls below the threshold.

status changes to *state*
Specifies the state of the server or component to monitor. If *state* changes to the specified value, the trigger executes. The state value is dependent on the object type. See Appendix C, "RMS Server and Component States" for information about the state codes.

wait *wait_interval*
Specifies the number of seconds to wait before triggering the event. This allows the object time to recover. If you do not include the wait option, the event triggers immediately.

continuous *continuous_flag*
A Boolean flag that, if set to true, causes the RMS to execute the trigger's script at every subsequent monitoring interval until the state changes. If you do not set this flag, the RMS executes the trigger script only once.

execute *command*
Specifies the command to be executed when the event is triggered. The command is operating-system-specific.

Examples

Example 1 Adds a trigger that executes the script *email.sh* when the status of the server named INVENTORY_RS is changed to "DOWN":

```
add status trigger for INVENTORY_RS
status changes to DOWN
execute /sybase/RMS/scripts/email.sh
```

Example 2 Adds a trigger that executes the script *email.sh* after 120 seconds. Since the status of the connection "inventory_pds.pdb1" of server INVENTORY_RS is changed to "SUSPENDED", it will execute script at every subsequent monitoring interval until the state changes:

```
add status trigger connection inventory_pds.pdb1 for
    INVENTORY_RS
status changes to Suspended
wait 120
continuous true
execute /sybase/RMS/scripts/email.sh
```

Example 3 Adds a trigger to the Replication Server INVENTORY_RS partition “p1” that executes the script *email.sh* when the partition usage exceeds 80 percent. The script is executed at every subsequent monitoring interval as long as the partition usage exceeds 80 percent:

```
add size trigger partition p1 for INVENTORY_RS
    size exceeds 80
    continuous true
    execute /sybase/RMS/scripts/email.sh
```

Example 4 Adds a trigger to the Replication Server INVENTORY_RS that executes the script *email.sh* when the sum of all partition usage exceeds 75 percent:

```
add size trigger partition for INVENTORY_RS
    size exceeds 75
    execute /sybase/RMS/scripts/email.sh
```

Example 5 Adds a trigger to the queue “inventory_pds.vendor(Inbound)” of Replication Server INVENTORY_RS that executes the script *email.sh* when the queue size falls below 100 megabytes. The script is executed at every subsequent monitoring interval as long as the queue size is less than 100 MB:

```
add size trigger queue inventory_pds.vendor(Inbound)
    for INVENTORY_RS
    size falls below 100
    continuous true
    execute /sybase/RMS/scripts/email.sh
```

Example 6 Adds a trigger to the replicate connection “inventory_rds.vendor” of replicate Replication Server INVENTORY_RS that will execute the script *email.sh* when the latency from the primary connection “inventory_pds.vendor” exceeds 5 minutes (300 seconds):

```
add latency trigger connection inventory_rds.vendor
    with primary inventory_pds.vendor
    for INVENTORY_RS
    latency exceeds 300
    execute /sybase/RMS/scripts/email.sh
```

Usage

- You can add one status trigger for each server or component status. For example, you can add a trigger for a Replication Server when the status changes to “DOWN” or “SUSPECT”, but you cannot add two triggers to the “DOWN” status.
- You must set *server_name* to the name of the replicate Replication Server when adding a latency connection trigger. In the following example, INVENTORY_RS is the replicate Replication Server:

```
add latency trigger connection inventory_rds.vendor
    with primary invetory_pds.vendor
    for INVENTORY_RS
    latency exceeds 300
     execute /sybase/RMS/scripts/email.sh
```

- You must set the configuration parameter ltl_origin_time_require to "true" when setting up a latency connection trigger where the primary connection is from a Replication Agent or MRA. To set the parameter, connect to the Replication Agent or MRA and execute:

```
ra_config ltl_origin_time_required, true
```

- add *event* trigger returns the following result set:

Table 9-2: Column descriptions for add event trigger

Column	Description
Action	The name of the action
Result	The result of the execution

See also

drop event trigger, get triggers

add *server*

Description

Adds a server to be monitored by the RMS.

Syntax

add {ASA | ASE | DirectConnect | IQ | Replication Agent | MRA | Replication Server | RMS | Open Server | dbltm} *server_name*
set username [to] *user*
[set password [to] *passwd*]
[set charset [to] *charset*]
[set language [to] *lang*]
[set rssd_username [to] *rssd_user*]
[set rssd_password [to] *rssd_passwd*]
[set rssd_charset [to] *rssd_charset*]
[set rssd_language [to] *rssd_lang*]
[set monitoring [to] {'true' | 'false'}]
[set interval [to] *interval*]
[set connection_ds [to] *ds*]
[set connection_db [to] *db*]

Parameters

ASA, ASE, DirectConnect, IQ, Replication Agent, MRA, Replication Server, RMS, Open Server, dbltm

Specifies the type of server to add to the RMS. You can add a remote RMS to a controlling RMS.

server_name
Specifies the name of the server as listed in the RMS *interfaces* file or LDAP server.

user
Specifies the user name that the RMS uses when establishing a connection to the server. The user name must have the required permissions to allow the RMS to monitor the server.

passwd
Specifies the corresponding password that the RMS uses when establishing a connection.

charset
Specifies the character set that the RMS uses when establishing a connection to the server. If you do not specify *charset*, jConnect uses the server's default character set.

lang
Specifies the language that the RMS uses when establishing a connection to the server. If you do not specify the language, jConnect uses the server's default language.

rssd_user
Specifies the user name that the RMS uses when establishing a connection to the server that contains the RSSD. The user name must have the required permissions to allow the RMS to monitor the server. This parameter is required for a Replication Server.

rssd_passwd
Specifies the corresponding password that the RMS uses when establishing a connection to the server that contains the RSSD.

rssd_charset
Specifies the character set that the RMS uses when establishing a connection to the server that contains the RSSD. If you do not provide the *charset*, jConnect uses the server's default character set.

rssd_lang
Specifies the language that the RMS uses when establishing a connection to the server that contains the RSSD. If you do not provide the language, jConnect uses the server's default language.

monitoring
Specifies whether the RMS is monitoring the state of the server and its components. If this value is false, monitoring for this server is disabled. If this value is true (the default), RMS automatically monitors this server.

interval

Specifies the number of seconds between monitoring cycles. If the monitoring property is set to true, then RMS performs periodic monitoring based on the value of *interval*. For example, if the value is set to 120, the RMS checks the health of the server every 120 seconds. The range of values is 30 seconds to 1 hour and the default value for the interval is the value of *ping_interval* in RMS config.

ds

Specifies the name of the primary data server. The dbltm sends *ds.db* to the Replication Server when replicating transactions. The *ds* must match the server name used in the Replication Server connection. This parameter is optional and is valid only for a dbltm server.

db

Specifies the name of the primary database. The dbltm server sends *ds.db* to the Replication Server when replicating transactions. The *db* must match the database name used in the Replication Server connection. This parameter is optional and is valid only for a dbltm server.

Examples

Example 1 Adds a Replication Server named INVENTORY_RS to the RMS. Uses the user name "sa" without a password, character set, or language when establishing a connection. Uses the user name "sa" and the password "sa_pwd" when establishing a connection to the RSSD:

```
add replication server INVENTORY_RS
    set username to sa
    set rssd_username to sa
    set rssd_password to sa_pwd
```

Example 2 Adds a server named INVENTORY_PDS to the RMS. Sets the user name, password, language, monitoring and interval:

```
add ASE INVENTORY_PDS
    set username to sa
    set password to sa_ps
    set language to Japanese
    set monitoring to true
    set interval to 120
```

Usage

- Use the RSSD options when adding a Replication Server to the RMS. You need not add the server that contains the RSSD to the RMS.
- The server name must be in the *interfaces* file or LDAP server that is used by the RMS.

- When you issue add *server*, the RMS attempts to connect to the specified server and automatically determines its type and version. If the type or version is invalid or cannot be determined, or the server is already being monitored, the RMS returns an error message.
- If the new server is a Replication Server, supply the user name for the RSSD.
- The add *server* command returns the following result set:

Table 9-3: Column descriptions for add server

Column	Description
Action	The name of the action
Result	The result of the execution

See also

configure server, connect to server, disconnect server, drop server, get servers, shutdown server

configure *component*

Description

Returns configuration parameter information for a component in either a Replication Server or an Adaptive Server; or sets the value of a specified configuration parameter. Components are monitored objects within a server. Replication Server components are connections, logical connections, and routes; Adaptive Server Enterprise components are RepAgent threads.

Syntax

```
configure {connections | logical connections | routes | repagents}
component_name
          [for] {server_name | group_name} [param[= value]]
```

Parameters

connections, logical connections, routes, repagents
: Specifies the type of component to configure. Replication Server components are connections, logical connections, routes; Adaptive Server Enterprise components are RepAgent threads.

component_name
: Specifies the name of the component to configure.

server_name
: Specifies the server that contains the requested component.

group_name
: Specifies the name of a group. You can modify the *group_name* parameter for each different component in the group.

param
Specifies the name of a component's configuration parameter.

value
The value to be assigned to the configuration parameter specified in the *param* option.

Examples

Example 1 Returns a list of all configuration parameters for the connection "inventory_pds.vendor" in the server INVENTORY_RS:

```
configure connection inventory_pds.vendor
    for INVENTORY_RS
```

Example 2 Returns the dsi_cmd_batch_size configuration parameter information for the connection "inventory_pds.vendor" in the server INVENTORY_RS:

```
configure connection inventory_pds.vendor
    for INVENTORY_RS dsi_cmd_batch_size
```

Example 3 Sets the dsi_cmd_batch_size configuration parameter to 15000 for the connection "inventory_pds.vendor" in the server INVENTORY_RS:

```
configure connection inventory_pds.vendor
    for inventory_rs dsi_cmd_batch_size = 15000
```

Usage

configure *component* returns the following result set if a *value* parameter is not included:

Table 9-4: Column descriptions for configure component

Column	Description
Server	The name of the server that contains the parameters.
Component Name	The name of the component that contains the parameter.
Component Type	The type of the component (connection, route, or RepAgent).
Category	The name of the category for the parameter. Categories are used to group related parameters together.
Parameter Name	The name of the parameter.
Current Value	The current value of the parameter.
Pending Value	The pending value becomes the value of the parameter after the component is restarted.
Default Value	The default value of the parameter.
Legal Values	A string that defines the legal values for the parameter. This can be a list, or a numeric range.
Restart Required	A flag indicating whether the server must be restarted for the parameter to take effect.

See also

get component, resume component, suspend component

configure *RMS*

Description

Returns configuration parameter information for the Replication Monitoring Services, or sets the value of a specified RMS configuration parameter.

Syntax

configure [*param* [= *value*]]

Parameters

param

Specifies the name of an RMS configuration parameter.

value

The value to be assigned to the configuration parameter specified in the *param* option.

Table 9-5 identifies all parameters for the RMS and their associated values:

Table 9-5: RMS parameters

Parameter	Value
Logconfig	The path to the RMS log config file.
Name	The name of the RMS server. This name must appear in the Sybase interfaces file.
Password	The password used to connect to the RMS. The value of this parameter is not displayed by the configure command.
ping_interval	The number of seconds between the end of one monitoring cycle and the beginning of the next. It ranges from 30 seconds to 3600 seconds.
Port	The IP port used by the RMS. It ranges from 1024 to 65,535.
SybaseHome	The Sybase home directory. This directory contains the interfaces file.
Username	The user name to connect to the RMS.
Version	The version string of the RMS. This is a read-only parameter.
includeLDAP	A flag that turns LDAP support on or off.
ldapTimeout	A user-configurable timeout value.

Examples

Example 1 Returns the list of RMS configuration parameters and their current value in this format:

```
configure


Parameter Name Parameter Type Current Value
-------------- -------------- ------------------
includeldap    boolean        false
ldaptimeout    integer        35
logconfig      string         ../plugins/
                              com.sybase.rms/
                              log4j.properties
name           string         RedtailRMS

Pending Value Default Value         Legal Values
------------- --------------------- ----------------
NULL          false                 List: true,false
NULL          180                   N/A
N/A           ../log4j.properties   N/A
NULL          Rms                   N/A

Category  Restart Required
--------  ----------------
Rms       false
Rms       false
Rms       N/A
Rms       true

Description
-----------------------------------------
A flag that turns LDAP support on or off.
A user configurable timeout value.
The path to the RMS log config file.
The name of the RMS server.

...
```

Example 2 Configures a user name of "sa" for the RMS:

```
configure username=sa
```

Usage

The configure *RMS* command returns this result set, if you do not include a value parameter:

Table 9-6: Default RMS result set

Column	Description
Parameter Name	The name of the parameter, such as `logconfig`, `name`, `port`, and `password`.
Parameter Type	The type of parameter, such as `boolean`, `integer`, `string`, and `password`.
Current Value	The current value of the parameter.
Pending Value	The value the parameter will be after the server is restarted.
Default Value	The default value of the parameter.
Legal Values	A string that defines the legal values for the parameter. This can be a list, or a numeric range.
Category	The name of the category for the parameter. You can use categories to group related parameters together.
Restart Required	A flag indicating whether the server must be restarted for the parameter to take effect.
Description	The parameter description.

See also

get version, resume Replication Agent, suspend Replication Agent, trace

configure server

Description

Returns configuration parameter information for a Replication Server or Replication Agent and Mirror Replication Agent (MRA), or sets the value of a specified configuration parameter. Also retrieves and sets RMS-specific parameters.

Syntax

configure server {*server_name* | *group_name*} [*RMS*] [*param* [= *value*]]

Parameters

server_name
Specifies the server to be configured.

group_name
Specifies the name of a group. Modify *group_name* for each server in the group.

RMS
Specifies RMS parameters.

param
Specifies the name of a server's configuration parameter.

value
The value assigned to the configuration parameter specified in the *param* option.

Examples

Example 1 Returns a list of all configuration parameters for the server INVENTORY_RS:

```
configure server INVENTORY_RS
```

Example 2 Returns the *memory_limit* configuration parameter information for the server INVENTORY_RS:

```
configure server INVENTORY_RS memory_limit


Parameter     Parameter  Current  Pending  Default
Name          Type       Value    Value    Value
------------  ---------  -------  -------  -------
memory_limit  NULL       20       55       NULL

Legal   Category  Restart    Description
Values            Required
------  --------  --------   -------------------------
NULL    NULL       NULL        NULL
```

Example 3 Sets the *memory_limit* configuration parameter to 50 for the server INVENTORY_RS:

```
configure server inventory_rs memory_limit = 50
```

Example 4 Retrieves all RMS-specific parameters:

```
configure server INVENTORY_RS RMS
```

Example 5 Changes the user name used by the RMS to connect to the server:

```
configure server INVENTORY_RS RMS username = 'rsa'
```

Usage

- configure server supports Replication Server, Replication Agent, and remotely monitored RMS configurations.
- configure server can retrieve and set RMS-specific parameters for all types of servers. The server and the RMS use these parameters to communicate.
- configure server returns the following result set if you do not include a value parameter:

Table 9-7: Default configure server result set

Column	Description
Parameter Name	The name of the parameter.
Parameter Type	The type of parameter.
Current Value	The current value of the parameter.
Pending Value	The pending value becomes the value of the parameter after the server is restarted.
Default Value	The default value of the parameter.
Legal Values	A string that defines the legal values for the parameter. This can be a list or a numeric range.
Category	The name of the category for the parameter. Categories are used to group related parameters together.
Restart Required	A flag indicating whether the server must be restarted in order for the parameter to take effect.
Description	The parameter description.

See also

add server, connect to server, disconnect server, drop server, get servers, shutdown server

connect to *server*

Description	Provides a pass-through mode that enables you to send commands to a server that is monitored by the RMS. The result sets generated by the commands are passed back to the client. You can connect to one server at a time to send commands.
Syntax	connect [to] *server_name* [username=*username* [,password = *pwd*]]
Parameters	*server_name* Specifies the name of the server to which to connect. *username* An optional parameter that specifies a user name to use when connecting to the server. If you omit this parameter, the RMS uses the name used when the server was added. *pwd* The password associated with the user name.
Examples	Establishes a connection to the server INVENTORY_RS: `connect to INVENTORY_RS`
Usage	• Issuing the connect command establishes a connection to the server. The message `Established a connection to the server` ***`server_name`*** indicates the connection is established. • Subsequent commands are passed directly to the server until the client issues a disconnect command. Use ISQL commands appropriate for the server; for example, Transact-SQL for Adaptive Server Enterprise, or RCL for Replication Server.
See also	add server, configure server, disconnect server, drop server, get servers, shutdown server

create group

Description	Defines a logical group of servers, and enables you to issue commands to the group.
Syntax	create group *group_name* [add] *server_name* [, *server_name*]
Parameters	*group_name* Specifies the name of the new group.

server_name
Specifies a server to add to the group.

Examples

Adds a group called "inventory_mra" that contains three Mirror Replication Agent (MRA) servers:

```
create group inventory_mra
    add ny_mra, chi_mra, la_mra
```

Usage

- A group name must be unique.
- All servers in a group must be the same type (that is, all servers must be MRAs, Replication Servers, and so on).
- A server can belong to more than one group.
- create group returns the following result set:

Table 9-8: Column descriptions for create group

Column	Description
Action	The name of the action
Result	The result of the execution, such as `Successfully created the group` *`group_name`*

See also

delete group, get group

delete group

Description

Deletes a logical group that was added using the create group command.

Syntax

delete group *group_name*

Parameters

group_name
Specifies the name of the group to delete.

Examples

Deletes the group named "inventory_mra:"

```
delete group inventory_mra
```

Usage

- Deleting a group does not drop the servers from the RMS.
- delete group returns the following result set:

Table 9-9: Column descriptions for delete group

Column	Description
Action	The name of the action
Result	The result of the execution, such as `Successfully dropped the group` *`group_name`*

See also

create group, get group

disconnect *server*

Description

Disconnects from a server where a pass-through connection was established. The client can connect through the RMS to a managed server using the connect command. Subsequent commands are forwarded to the server until the client issues the disconnect command.

Syntax

disconnect

Examples

From the client, disconnects from a server:

```
disconnect
```

Usage

Issuing the disconnect command breaks the connection to the server. The message `Disconnected from the server` ***`servername`*** indicates the connection no longer exists.

See also

add server, configure server, connect to server, drop server, get servers, shutdown server

drop *event* trigger

Description

Removes a trigger that the RMS is monitoring. A trigger identifies a process or script that is executed by the RMS. Set triggers up using the add trigger command.

Syntax

drop {status | latency | size} trigger
    [{connection | logical connection | route | queue | rep agent |
      partition} [*component_name*]]
    [with primary *primary_connection*]
    for *server_name*
    {status changes to *state* |

size {exceeds | falls below} *size_threshold* |
latency {exceeds | falls below} *latency_threshold*}

Parameters

status, latency, size
Specifies the type of trigger.

connection, logical connection, route, queue, rep agent, partition
Specifies the type of component.

component_name
Specifies the name of the component. Components are monitored objects within a server. Replication Server components are connections, logical connections, routes, queues, and partitions; Adaptive Server Enterprise components are RepAgent threads.

with primary *primary_connection*
Identifies the primary connection of the latency connection trigger to drop. This parameter is required when dropping a latency connection.

server_name
Specifies the name of the server for which the trigger is defined that is being dropped.

state
Specifies the state of the event trigger that is being dropped. See Appendix C, “RMS Server and Component States” for state information.

size exceeds, falls below *size_threshold*
Indicates the size trigger to drop.

latency exceeds, falls below *latency_threshold*
Indicates the latency trigger to drop.

Examples

Example 1 Removes the “DOWN” status trigger for the server INVENTORY_RS:

```
drop status trigger for INVENTORY_RS
    status changes to DOWN
```

Example 2 Removes the “SUSPENDED” status trigger for the connection “inventory_pds.pdb1” of server INVENTORY_RS:

```
drop status trigger connection inventory_pds.pdb1
    for inventory_rs
    status changes to SUSPENDED
```

Example 3 Drops a partition size trigger:

```
drop size trigger partition p1
   for INVENTORY_RS
   size exceeds 80
```

Example 4 Drops a latency connection trigger:

```
drop latency trigger
   connection inventory_rds.vendor
   with primary inventory_pds.ventory
   for INVENTORY_RS
   latency exceeds 300
```

Usage

drop trigger returns the following result set:

Table 9-10: Column descriptions for drop event trigger

Column	Description
Action	The name of the action
Result	The result of the execution

See also

add event trigger, get triggers

drop server

Description

Drops a server that is being monitored by the RMS.

Syntax

drop server *server_name*

Parameters

server_name
Specifies the name of the server to be removed from the RMS.

Examples

Drops the server named INVENTORY_RS from the RMS. The agent no longer monitors the server:

```
drop server inventory_rs
```

Usage

drop server returns the following result set:

Table 9-11: Column descriptions for drop server

Column	Description
Action	The name of the action
Result	The result of the execution

See also

add server, configure server, connect to server, disconnect server, get servers, shutdown server

filter *connection*

Description

Returns current filter settings, or sets the filter setting for a connection. The command can filter either the Replication Agent thread or the DSI thread status.

Syntax

filter *connection* for *replication_server_name* [{rep agent | dsi} [={on | off}]]

Parameters

connection
Specifies the name of the connection to filter.

replication_server_name
The name of the Replication Server to filter.

rep agent, dsi
Specifies the part of the connection to filter.

on, off
Sets filtering for the connection to either on or off.

Examples

Example 1 Returns the list of filter set for "inventory_pds.vendor" connection in prs1:

```
filter inventory_pds.vendor for prs1
```

Example 2 Hides the status of the DSI thread for the connection "inventory_pds.vendor" in prs1:

```
filter inventory_pds.vendor dsi on for prs1 dsi = on
```

Example 3 Turns rep agent filtering off for the connection "inventory_pds.item" in prs1:

```
filter inventory_pds.item off for prs1 rep agent = off
```

Usage

- When a filter is turned on, the connection status is displayed as "Hidden." The status of the connection is not rolled up into the status of the Replication Server.
- If the rep agent filter is turned on, the RMS does not report the status of the Replication Agent thread or RepAgent thread in the Adaptive Server Enterprise, Replication Agent, or the Replication Server.
- When you invoke the filter command with no options specified, it returns a list of specified connections.
- filter returns the following result set:

Table 9-12: filter connection result set (list of filtered connections)

Column	Description
RepServer	The name of the Replication Server
Connection	The name of the connection
DSI	The filtering value of DSI
rep agent	The filtering value of rep agent

- The filter command returns the following result set, if you have turned filtering on or off for the connection:

Table 9-13: filter connection result set (filtering turned on/off)

Column	Description
Action	The name of the action
Result	The result of the execution

See also

get network spec, get threads

get *component*

Description

Returns a list of components that are monitored by the RMS. Components are monitored objects within a server. Replication Server components are connections, logical connections, routes, queues, and partitions; Adaptive Server Enterprise components are RepAgent threads.

Syntax

```
get {connections | logical connections | routes | queues | partitions |
        repagents}
    for server_name [,component_name ]...
```

Parameters

connections, logical connections, routes, queues, partitions, repagents
: Returns the specified type of component monitored by the RMS. For example, returns all connections in a specified Replication Server monitored by the RMS.

server_name
: Specifies the server that contains the requested components. If the server does not contain any of the requested components, get *component* returns an empty result set.

component_name
Specifies a specific component or list of components to return. Components are monitored objects within a server. Replication Server components include connections, logical connections, routes, queues, and partitions. Adaptive Server Enterprise components are RepAgent threads.

Examples

Example 1 Returns a list of all connections being monitored by the RMS in the Replication Server INVENTORY_RS:

```
get connections for INVENTORY_RS
```

Example 2 Returns a list of all RepAgent threads being monitored by the RMS in the Adaptive Server Enterprise server called INVENTORY_PDS:

```
get repagents for INVENTORY_PDS
```

Example 3 Returns the information for the route named “inventory_rs.euro_sales” for the Replication Server INVENTORY_RS:

```
get routes for INVENTORY_RS, inventory_rs.euro_sales
```

Usage

- Components monitored by a remote RMS are also returned by this command.
- get connections supports retrieving connections that are associated with a data server or a Replication Agent process. It supports servers other than a Replication Server:
  - ASE – get connections returns the connection information for each database in the ASE. The RMS searches all of the Replication Servers in the RMS looking for connections named *ASE_name.database*.
  - Replication Agent/MRA – get connections returns the information for the primary connection associated with the Replication Agent. The name of the connection associated with the Replication Agent or MRA is stored in the configuration parameters rs_source_ds and rs_source_db. get connections searches all of the Replication Servers in the RMS to find the connection.
  - dbltm – get connections returns the information for the primary connection associated with the dbltm. The connection information for the dbltm is optionally provided when the server is added to the environment. If the information is not available, get connections returns an empty result set and writes a warning message to the RMS log indicating the information is missing.
  - DirectConnect – get connections returns the information of all of the connections where the data server matches the name of the DirectConnect server.

    - ASA/IQ – get connections returns the information where the data server matches the name of the ASA or IQ server. ASA or IQ server does not use database names.

- If the specified server is not monitored by the RMS, the get *component* command returns an error message.
- get *component* returns the following result set (some results vary by component type):

Table 9-14: Column descriptions for get component result set

Column	Description
Server	The name of the server that contains the components.
Name	The name of the component.
Type	The type of the component (connection, route, queue, RepAgent).
Last Monitored	A timestamp indicating that last time the component was monitored by the RMS. The timestamp is in the format MM/DD/YYYY HH:MM:SS.
State	The description that defines the state of the component.
State Constant	The integer constant that defines the state of the component. See Appendix C, “RMS Server and Component States” for state information.
Description	The reason string that describes the state of the component.
More Descriptions	Indicates whether additional information is available. If true, then the status of the component contains multiple descriptions. Use the get status descriptions command to retrieve a list of all descriptions for the component.
Intermediate RepServer	Identifies the intermediate site for the route. Intermediate RepServer should be blank if the route is a direct route
Queue Number	The queue number.
Queue Type	The queue type.
Size column	The queue size.

See also

configure component, get status descriptions, get servers, resume component, suspend component

get group

Description

Returns a result set that contains either a list of the groups and a roll-up status for each group, or status of each server in a group and a roll-up status for the specified group. Roll-up status shows the lowest status reported; for example, if any server in a group is not UP, then the group status is reported as "SUSPECT".

Syntax

get group [*group_name*]

Parameters

group_name

Specifies the name of the group for which to retrieve the list of servers.

Examples

Example 1 Returns a list of the groups names, and a roll-up status for each group:

```
                    get group
Group Name  State       State     Description               More
            Constant                                        Descriptions
----------  --------    -------   ------------------------  ------------
group1      4           Suspect   inventory_rs1 is Suspect  False
```

Example 2 Returns the status of each list of server names that the group "inventory_mra" contains and a roll-up status for the group:

```
                    get group inventory_mra

Group Name     Server Name  Server Type         Last Monitored
-------------  -----------  ------------------  --------------------
inventory_mra  RAObeta      Replication Agent    12/16/2005 13:38:30

Version String
---------------------------------------------------------------------------
Sybase Replication Agent for Unix & Windows/12.6.0.5001/B/generic/
JDK 1.4.2/main/5001/VM: Sun Microsystems Inc. 1.4.2_05/OPT/Wed May  4
02:42:07 MDT 2005

State Constant  State   Description                    More Descriptions
--------------  ------  -----------------------------  -------------------
6               Admin   Waiting for operator command.  false
```

Usage

- If you do not provide a *group_name* parameter, get group returns a result set that contains a roll-up status for each group:

Table 9-15: Column descriptions for get group (group list, and roll-up for each group)

Column	Description
Group Name	The name of the group.
State Constant	The integer constant that defines the state of the group.
State	The description that defines the state of the group. This is a string representation of the State Constant column.
Description	The reason string that describes the state of the group. If there is more than one description, this field should contain the first description.
More Descriptions	A flag that indicates whether there is more than one description string that describes the status of the group.

- If you provide a *group_name* parameter, get group returns a result set that contains the status of each server:

Table 9-16: Column descriptions for get group (individual server, and roll-up for specified group)

Column	Description
Group Name	The name of the group.
Server Name	The name of the server.
Server Type	The type of the server (Replication Server, Adaptive Server Enterprise, Replication Agent, and so on).
Last Monitored	A timestamp indicating that last time the server was monitored by the RMS. The timestamp is in the format *MM/DD/YYYY HH:MM:SS*.
Version String	Returns the server version string.
State Constant	The numeric status of the server.
State	The description that defines the state of the server. This is a string representation of the state constant.
Description	The reason string that describes the state of the server.
More Descriptions	Indicates whether additional information is available. If true, then the status of the server contains multiple descriptions. Use get status descriptions to retrieve a list of all descriptions for the server.

See also

create group, delete group, get status descriptions

get heartbeat

Description

Retrieves the heartbeats that have been defined in the RMS. A heartbeat is a process that runs the Replication Server rs_ticket stored procedure at the primary database at a specified interval. The output, or heartbeat ticket, is stored in a table in the replicate database.

Syntax

get heartbeat [for *ds.db*]

Parameters

ds.db
The name of a connection that is participating in a heartbeat process. This name can be either a primary or replicate connection.

Examples

Example 1 Retrieves all heartbeats defined in the RMS:

```
get heartbeat
```

Example 2 Retrieves heartbeats defined for the “inventory_pds.pdb1” connection:

```
get heartbeat for inventory_pds.pdb1
```

Usage

get heartbeat returns the following result set:

Table 9-17: Column descriptions for Get Heartbeat

Column	Type	Description
Primary	varchar	The name of the primary data server and database.
Replicate	varchar	The name of the replicate data server and database.
Interval	int	The interval in seconds that the RMS executes the rs_ticket command.
Max Rows	int	The maximum number of rows that the rms_ticket_history table can contain. The RMS tests the size of the table at every heartbeat interval. If the size is greater then *max_rows*, the RMS removes the oldest entries.

See also

get heartbeat tickets, start heartbeat, stop heartbeat

get heartbeat tickets

Description

Retrieves a set of tickets from the rms_ticket_history table for the heartbeat process and date and time range specified. The ticket output includes a set of date and time fields for each step in the replication process. The date and time are synchronized to the replicate data server system time.

Syntax

get heartbeat tickets from *pds.pdb* to *rds.rdb*
    [start *date time*]
    [end *date time*]
    [last *num_tickets*]

Parameters

pds.pdb
The name of the primary data server and database.

rds.rdb
The name of the replicate data server and database.

start *date time*
The starting date and time for the range of tickets. The RMS retrieves ticket information starting with this time and ending at either the end time, or the end of the table. If you do not provide this parameter, the RMS starts at the oldest ticket in the table.

end *date time*
The ending date and time for the range of tickets. The RMS retrieves ticket information starting at the specified time until this time. If you do not provide this parameter, the RMS includes all tickets starting with the start time.

last *num_tickets*
Retrieves the specified number of tickets from the table. You cannot use this parameter with the start and end parameters

Examples

Example 1 Retrieves all rows from the rms_ticket_history table:

```
get heartbeat tickets
    from inventory_pds.vendor to inventory_dss.vendor
```

Example 2 Retrieves all rows between Oct 29th and November 3rd:

```
get heartbeat tickets
    from inventory_pds.vendor to inventory_dss.vendor
    start Oct 29, 2005 12:00am
    end Nov 3, 2005 12:00am
```

Example 3 Retrieves all rows in the table starting at October 29th at 1:30:

```
get heartbeat tickets
    from inventory_pds.vendor to inventory_dss.vendor
    start 10/29 1:30pm
```

Example 4 Retrieves the 500 latest rows in the table:

```
get heartbeat tickets
    from inventory_pds.vendor to inventory_dss.vendor
    last 500
```

Usage

- The start and end parameters support multiple date and time formats; for example, you can enter the date in the format MM/DD/YYYY (such as 10/29/2005), or in the format MMM DD, YYYY (such as Oct 29, 2005). The time fields support an entry without seconds or milliseconds, as well as localized date and time formats.
- All dates in the result set are synchronized to the replicate data server system time. Before the result set is generated, the RMS retrieves the date and time from the data servers and Replication Servers, and adjusts the time by the difference between the server's time and the RMS system's time.
- The get heartbeat ticket command returns the following result set:

Table 9-18: Column descriptions for get heartbeat tickets

Column	Type	Description
Primary	varchar	The name of the primary data server and database.
Replicate	varchar	The name of the replicate data server and database.
PDB	datetime	The time that the rs_ticket stored procedure was executed at the primary database.
EXEC	datetime	The time the ticket passed through the primary Replication Server executor thread.
Bytes	int	Total bytes the executor thread received from the RepAgent or Replication Agent.
DIST	datetime	The time the ticket passed through the primary Replication Server distributor thread.
DSI	datetime	The time the ticket passed through the replicate Replication Server DSI thread.
RDB	datetime	The time the ticket arrived at the replicate data server. The result set is sorted by the RDB field.

See also

get heartbeat, start heartbeat, stop heartbeat

get network spec

Description

Retrieves the connection information for all servers known to the RMS. This list is retrieved from the RMS *interfaces* file or LDAP server. The list consists of the server name, host computer name, and the port number used by the server.

Syntax

get network spec [[monitored] | [*server_name* [,*server_name*]]]

Parameters

monitored
Returns the list of servers that the RMS is currently monitoring.

server_name
Specifies the name of a server or set of servers for which to retrieve information.

Examples

Example 1 Retrieves a list of all servers from the RMS *interfaces* file or LDAP server:

```
get network spec
```

Example 2 Retrieves the connection information for the set of servers managed by the RMS:

```
get network spec monitored
```

Example 3 Retrieves the connection information for the servers INVENTORY_RS and INVENTORY_ASE:

```
get network spec INVENTORY_RS, INVENTORY_ASE
```

Usage

- Returns an empty result set if the requested server does not exist or the *interfaces* file or LDAP server is not available.
- get network spec returns the following result set:

Table 9-19: Column descriptions for get network spec

Column	Description
Name	The name of the server
Host	The name of the computer that hosts the server
Port	The port number of the host on which the server listens

See also

filter connection

get rmiaddress

Description

Retrieves the address of Remote Method Invocation (RMI) service. RMI enables an object running in one Java virtual machine (VM) to invoke methods on an object running in another Java VM. RMI provides remote communication between programs written in Java.

RMS provides client applications the ability to register callback routines that are executed when a specific event occurs. The RMS provides asynchronous callbacks using the remote RMI feature.

Syntax

get rmiaddress

Parameters

rmiaddress
Returns the server and port used for RMI service.

Examples

Retrieves the address of the RMI service:

```
get rmiaddress


Rmi Address
--------------------
rmi://redtail:9999/
```

Usage

get rmiaddress returns the address of the RMI service.

get servers

Description

Returns the status for each of the servers that are monitored by the RMS, followed by the status of the RMS environment. The RMS status is a roll-up of the monitored servers, and shows the lowest status reported; for example, if the status of any server in the list is not “UP”, then the status for the RMS is reported as “SUSPECT”.

Syntax

get servers [[for group *group_name*] | [{ASA | ASE | DirectConnect | IQ | Replication Agent | MRA | Replication Server | RMS | Open Server | [*server_name*,]…}]]

Parameters

ASA, ASE, DirectConnect, IQ, Replication Agent, MRA, Replication Server, RMS, Open Server
Returns only the specified type of server monitored by the RMS. For example, returns all Replication Servers monitored by the RMS.

group_name
Specifies a group for which servers are returned.

server_name
Specifies a specific server or list of servers to return. If the server is not monitored by the RMS, an empty result set is returned.

Examples

Example 1 Returns the status for all servers monitored by the RMS, followed by the status for the RMS environment:

```
get servers
```

Example 2 Returns a list of all Adaptive Server Enterprise servers monitored by the RMS:

```
get servers ASE
```

Example 3 Returns a list that contains the information for the servers INVENTORY_RS and INVENTORY_PDS”

```
get servers INVENTORY_RS, INVENTORY_PDS
```

Usage

Servers monitored by a remote RMS are also returned by this command.

Table 9-20: Column descriptions for get servers

Column	Description
Name	The server name.
Type	The server type (Replication Server, Adaptive Server Enterprise, Replication Agent, and so forth).
Last Monitored	A timestamp indicating that last time the server was monitored by the RMS. The timestamp is in the format *MM/DD/YYYY HH:MM:SS*.
Version String	The complete version string of the server.
State Constant	The integer constant that defines the state of the server. See Appendix C, “RMS Server and Component States” for server state information.
State	The description that defines the state of the server. This is a string representation of the state constant.
Description	A string that describes the state of the server.
More Descriptions	Indicates whether additional information is available. If true, then the status of the server contains multiple descriptions. Use the get status descriptions command to retrieve a list of all descriptions for the server.

See also

add server, configure server, connect to server, disconnect server, drop server, get component, get status descriptions, shutdown server

get status descriptions

Description

Retrieves the list of status descriptions for a server or component. Components are monitored objects within a server. The state of a server or component consists of a state integer constant and a list of description strings. The get server and get component commands return the first description in the list and a flag that indicates whether the description list contains more than one string.

Client applications can use get server or get component to display the state of all servers monitored by the RMS. If more information is needed, the application can display all descriptions.

Syntax

```
get status descriptions { [for {connection | logical connection | route | queue |
                rep agent | partition}
          component_name] for server_name | for group_name }
```

Parameters

connection, logical connection, route, queue, rep agent, partition
Returns status descriptions for the specified server or component.

component_name
Specifies the name of the component for which to return status descriptions. Components are monitored objects within a server. Replication Server components are connections, logical connections, routes, queues, and partitions. Adaptive Server Enterprise components are RepAgent threads.

server_name
Specifies the name of the server for which to return status descriptions. The server name is also used when returning status descriptions for components.

group_name
Specifies the name of the group for which to return status descriptions.

Examples

Example 1 Retrieves all description strings for the server name INVENTORY_RS:

```
get status descriptions for INVENTORY_RS
```

Example 2 Retrieves all description strings for the group name “group1”:

```
get status descriptions for group1
```

Example 3 Retrieves all description strings for the connection “inventory_pds.pdb1” in the server INVENTORY_ASE:

```
get status descriptions
    for connection inventory_pds.pdb1 for INVENTORY_ASE
```

Usage

- get status descriptions returns all strings in the description list (including the first description).

- You can use get status descriptions to return the status descriptions for the RMS.
- get status descriptions returns a result set that contains a single string column that contains one status description. The result set returns multiple rows, one for each description.

See also

get component, get servers

get threads

Description

Displays information about threads running in the Replication Server.

Syntax

get threads [for] *server_name* [{dist | dsi | rsi | sqm | sqt}]

Parameters

server_name
Specifies the Replication Server that contains the threads.

dist | dsi | rsi | sqm | sqt
Specifies the thread type. If no type is specified, the summary list of threads is returned.

Examples

Example 1 Returns the summary list of all threads in the Replication Server INVENTORY_RS:

```
get threads for INVENTORY_RS
```

Example 2 Returns the thread information for all route threads in the Replication Server INVENTORY_RS:

```
get threads for INVENTORY_RS rsi
```

Usage

get threads executes the admin who command for the specified Replication Server. The result set is identical to the admin who result set.

See also

filter connection, resume component, suspend component

get triggers

Description

Displays information about the triggers that are monitored by the RMS.

Syntax

```
get status triggers
          [{connection | logical connection | route | queue | rep agent |
```

partition}
component_name for *server_name*]

Parameters

status
Specifies the type of trigger.

connection, logical connection, route, queue, rep agent, partition
Specifies the type of component to be monitored. Components are monitored objects within a server. Replication Server components are connections, logical connections, routes, queues, and partitions. Adaptive Server Enterprise components are RepAgent threads.

component_name
Specifies the name of the component to be monitored.

server_name
Specifies the name of the server to be monitored.

Examples

Example 1 Returns the list of all triggers in the RMS:

```
get triggers
```

Example 2 Returns the list of all triggers defined for the Replication Server INVENTORY_RS:

```
get triggers for INVENTORY_RS
```

Example 3 Returns the list of all triggers defined for the connection "inventory_pds.vendor" in the Replication Server INVENTORY_RS:

```
get triggers connection inventory_pds.vendor for
    INVENTORY_RS
```

Usage

get triggers returns the following result set:

Table 9-21: Column descriptions for get triggers

Column	Description
Type	The type of the trigger.
Server Type	The server type of the trigger.
Server Name	The server name of the trigger.
Component Type	The component type of the trigger.
Component Name	The component name of the trigger.
Primary Connection	The name of the primary connection.
Change Value	The value of the server or component that will cause the RMS to execute the trigger's script.
Change State	The state string of the server or component that will cause the RMS to execute the trigger's script.
Wait	The number of seconds to wait after the initial state change before executing the trigger's script. If *waitInterval* is set to zero, the script executes immediately.
Continuous	A Boolean flag that, if set to true, causes the RMS to execute the trigger's script at every subsequent monitoring interval until the state changes. If the flag is not set, then the RMS executes the trigger script only once.
Script	The operating system script that the RMS executes when the event occurs.

See also

add event trigger, drop event trigger

get version

Description

Retrieves the version string of RMS.

Syntax

get version

Parameters

version

Returns a string containing several pieces of version information separated by slashes.

Examples

Retrieves the version string of the RMS:

```
    version
---------------------------------------------------------------------------
Replication Monitoring Services/15.0/P/generic/JDK 1.4.2.03/main/
```

```
    Build 102/VM:
    Sun Microsystems Inc. 1.5.0_05/Opt/Wed Dec 7 15:26:13 CST 2005
```

Usage	get version returns the version string of the RMS.
See also	configure RMS, resume Replication Agent, suspend Replication Agent, trace

log level

Description — Returns the current log level setting. log level also changes log level settings of RMS.

Syntax — log level [= {debug | info | warn | error | fatal}]

Parameters — debug, info, warn, error, fatal

The log level value.

Examples — **Example 1** Returns the current log level setting:

```
log level
```

Example 2 Sets the log level to error:

```
log level = error
```

Usage — The log level has the following order: debug, info, warn, error, fatal. You must set the log level to at least info to trace log level messages.

resume component

Description — Resumes a component in a specified server. The command resumes a DSI thread, Replication Agent thread, queue, or route in a Replication Server, or a RepAgent thread in an Adaptive Server Enterprise.

Syntax — resume {dsi | queue | rep agent | route} *component_name*
for {*server_name* | *group_name*} [skip transaction | execute transaction]

Parameters — dsi, queue, rep agent, route

Specifies the component type to resume. The component is a database name, if resuming a RepAgent thread in an Adaptive Server Enterprise. Otherwise, the component is a connection, queue, or route name.

component_name

Specifies the name of the component to resume.

group_name
Specifies the name of a group. Each component in the group is resumed.

server_name
Specifies the name of either a Replication Server or an Adaptive Server Enterprise that contains the component.

skip transaction
If the option is provided for a DSI connection, instructs the Replication Server to resume execution with the second transaction in the connection's queue. The first transaction is written to the database exceptions log.

If the option is provided for a queue, specifies that the SQM should skip the first large message encountered after restarting.

If this option is provided for a route, ignore the first transaction encountered with a wide message greater than 16K bytes.

execute transaction
Overrides the Replication Server restriction against the application of system transactions after a DSI start-up if the system transaction is the first transaction in the DSI queue.

Examples

Example 1 Resumes the DSI thread for the connection "inventory_pds.vendor" in the Replication Server INVENTORY_RS. Does not wait for the current operation to complete:

```
resume dsi inventory_pds.vendor for INVENTORY_RS with
       nowait
```

Example 2 Resumes the Replication Agent thread for the connection "inventory_pds.vendor" in the Replication Server INVENTORY_RS:

```
resume rep agent inventory_pds.vendor for INVENTORY_RS
```

Example 3 Starts the RepAgent thread for the database vendor in the Adaptive Server Enterprise INVENTORY_PDS:

```
resume rep agent vendor for INVENTORY_PDS
```

Usage

- The rep agent component type is used to resume either a Replication Agent thread for a connection in a Replication Server, or a RepAgent thread in an Adaptive Server Enterprise.
- The skip transaction option is valid with a Replication Server DSI connection, queue, or route.
- The execute transaction option is valid only for a Replication Server DSI connection.resume issues the sp_start_rep_agent when resuming a RepAgent thread in an Adaptive Server Enterprise.

- resume returns the following result set.

Table 9-22: Column descriptions for resume component

Column	Description
Action	The name of the action
Result	The result of the execution

See also

configure component, get component, get threads, suspend component

resume *Replication Agent*

Description

Resumes replication in a Replication Agent.

Syntax

resume {*server_name* | *group_name*}

Parameters

server_name
Specifies the name of the Replication Agent to resume.

group_name
Specifies the name of a group. Each Replication Agent in the group is resumed.

Examples

Resumes the Replication Agent “sales_ra:”

```
resume sales_ra
```

Usage

None

See also

configure RMS, get version, suspend Replication Agent, trace

shutdown *server*

Description

Issues a shutdown command to a server.

Syntax

shutdown {*server_name* | *group_name*} [*with nowait*]

Parameters

server_name
Specifies the server to be shut down.

group_name
Specifies the name of a group. Each server in the group is shut down.

with nowait
: Shut down the server immediately without waiting for the executing operation to complete.

Examples

Issues the shutdown command to the server named INVENTORY_RS:

```
shutdown INVENTORY_RS
```

Usage

The RMS allows the user to shut down *only* Replication Server, Replication Agent, and Mirror Replication Agent.

See also

add server, configure server, connect to server, disconnect server, drop server, get servers

start heartbeat

Description

Sets up and starts a heartbeat process from a specified primary connection to a specified replicate connection.

Syntax

```
start heartbeat from pds.pdb to rds.rdb
          [set interval [to] hb_interval]
          [set maximum rows [to] max_rows]
          [do not load rs_ticket_report]
```

Parameters

pds.pdb
: The name of the primary data server and database. The name must be associated with an existing primary connection.

rds.rdb
: The name of the replicate data server and database. The name must be associated with an existing primary and replicate, or replicate-only connection.

hb_interval
: The interval in seconds that the RMS executes the rs_ticket command. The default is 60 seconds.

max_rows
: The maximum number of rows that the rms_ticket_history table can contain. The RMS tests the size of the table at every heartbeat interval. If the size is greater then *max_rows*, the RMS removes the oldest entries. The RMS deletes 10% of the *max_row* size rows in the table. The default is 5000 rows.

do not load rs_ticket_report
If this flag is included, the RMS does not load the rs_ticket_report and you can provide a custom stored procedure instead. You must provide an rs_ticket_report procedure that loads the rms_ticket_history table with the required information.

Examples

Sets up and starts the heartbeat process, then executes the rs_ticket procedure every 60 seconds; limits the rms_ticket_history table to 5000 rows:

```
start heartbeat
      from inventory_pds.vendor to inventory_dss.vendor
```

Usage

- To set up the heartbeat, the RMS uses the user name that was provided when the server was added to the domain. The user names must have the correct permissions to create the table and stored procedure at the replicate database, configure the DSI at the replicate Replication Server, and execute the rs_ticket stored procedure at the primary database.
- The RMS can create only one heartbeat between a primary and replicate database. The RMS generates an error if a heartbeat already exists.
- The RMS does not delete an rms_ticket_history table if one already exists, but assumes that another heartbeat from a different primary database is already executing.
- The RMS assumes that the replicate database is set-up to receive data from the Replication Server and it neither checks for subscriptions nor generates a new one. Replication Server version must be at least 12.6.
- The Replication Server requires that the replicate database must have at least one subscription against a table, stored procedure, or database before the replicate Replication Server sends the rs_ticket information. The subscription does not have to be against any specific table or stored procedure. In case there is no subscription, rs_ticket functions in a warm-standby environment.

See also

get heartbeat, get heartbeat tickets, stop heartbeat

stop heartbeat

Description

Stops the heartbeat process between the primary and replicate databases. Optionally, truncates the rms_ticket_history table.

Syntax

stop heartbeat from *pds.pdb* to *rds.rdb*
    [delete history]

Parameters

pds.pdb
The name of the primary data server and database.

rds.rdb
The name of the replicate data server and database.

delete history
If included, the rms_ticket_history table is deleted when the heartbeat is stopped. By default, the table is not deleted.

Examples

Stops the heartbeat process:

```
stop heartbeat
     from inventory_pds.vendor to inventory_dss.vendor
```

Usage

Optionally, you can delete the rms_ticket_history table when the heartbeat is stopped. This means you can no longer retrieve tickets from the table.

See also

get heartbeat, get heartbeat tickets, start heartbeat

suspend *component*

Description

Suspends a component in a specified server. The command suspends a DSI thread, a route in a Replication Server, or a RepAgent thread in Adaptive Server Enterprise.

Syntax

suspend {dsi | rep agent | route} *component_name*
for {*server_name* | *group_name*} [*with nowait*]

Parameters

dsi, rep agent, route
Specifies the component type to suspend.

component_name
Specifies the name of the component to suspend. The component is a database name if you are suspending a RepAgent thread in an Adaptive Server Enterprise. Otherwise, the component is a connection or route name.

server_name
Specifies the name of either a Replication Server or an Adaptive Server Enterprise that contains the component.

group_name
Specifies the name of a group. Each component in the group is suspended.

with nowait
: Suspends the component immediately without waiting for the executing operation to complete.

Examples

Example 1 Suspends the DSI thread for the connection "inventory_pds.vendor" in the Replication Server INVENTORY_RS, without waiting for the current operation to complete:

```
suspend dsi inventory_pds.vendor
    for INVENTORY_RS with nowait
```

Example 2 Suspends the Replication Agent thread for the connection "inventory_pds.vendor" in the Replication Server named INVENTORY_RS:

```
suspend rep agent inventory_pds.vendor for INVENTORY_RS
```

Example 3 Stops the RepAgent thread for the database vendor in the Adaptive Server Enterprise named INVENTORY_PDS:

```
suspend rep agent vendor for INVENTORY_PDS
```

Usage

- The rep agent component type is used to suspend either a Replication Agent thread for a connection in a Replication Server, or a RepAgent thread in an Adaptive Server Enterprise.
- The with nowait option is valid with a Replication Server DSI connection or an Adaptive Server Enterprise RepAgent thread.
- suspend *component* issues the sp_stop_rep_agent stored procedure when suspending a RepAgent thread in an Adaptive Server Enterprise.
- suspend *component* returns the following result set:

Table 9-23: Column descriptions for suspend component

Column	Description
Action	The name of the action.
Result	The result of the execution.

See also

configure component, get component, get threads, resume component

suspend *Replication Agent*

Description

Suspends replication in a Replication Agent.

Syntax

suspend {*server_name* | *group_name*}

Parameters

server_name
Specifies the name of the Replication Agent to suspend.

group_name
Specifies the name of a group. Each Replication Agent in the group is suspended.

Examples

Suspends the Replication Agent “sales_ra”:

```
suspend sales_ra
```

Usage

None

See also

configure RMS, get version, resume Replication Agent, trace

trace

Description

Displays trace information in the RMS log file.

Syntax

trace [*flag* | all {on | off}]

Parameters

flag
Specifies the trace flag name for which you want to change settings.

all
A keyword that allows you to apply a switch value to all trace flags.

on, off
Indicates whether to enable or disable tracing for the trace point specified in the flag option.

Examples

Example 1 Returns the current settings for all RMS trace flags:

```
trace
```

Example 2 Turns the RMS_Command trace flag on:

```
trace RMS_Command on
```

Example 3 Turns off all trace flags:

```
trace all off
```

Usage

- The trace command should only be used by knowledgeable users to troubleshoot RMS.
- When trace is invoked with no options specified, it returns the current settings for all RMS trace flags.
- When trace is invoked with the flag and on, off options, it changes the setting of the trace point specified in the flag option.
- Changes made with the trace command take effect immediately.
- These trace flags are supported by RMS:

Table 9-24: Trace flags

Flag	Description
Add_Drop_Server	Write a message to the log when a server is added or dropped.
Add_Drop_Trigger	Write a message to the log when a trigger is added or dropped.
Client_Connection	Display information about a connection when a client initially connects to the RMS.
Configuration	Write a trace message to the log every time an RMS configuration parameter is changed.
Filter_Conn	Writes a trace message to the log when a connection is filtered.
Monitoring	Add trace messages to the RMS at each step of the monitoring cycle, and write a message before monitoring each server.
Network_Connection	Add trace messages to the RMS whenever a connection to a server is created. Include all connection information (except the password) in the trace message.
RMS_Command	Write every command received by the RMS to the error log.
Server_Command	Write every command sent to a monitored server by the RMS to the error log.
Shutdown_Server	Write a message to the log when the server is shut down.
Start_Stop_Heartbeat	Write a message to the log when a heartbeat is started or stopped.
Startup	Add trace messages to the RMS at each step of the start-up process.
Status_Change	Display server and component result description when status changes.
Suspend_Resume_Component	Write a message to the log when a component is suspended or resumed.
Trigger_Execution	Display message stating that event trigger was executed.

See also

configure RMS, get version, resume Replication Agent, suspend Replication Agent

APPENDIX A Acronyms and Abbreviations

This appendix lists acronyms and abbreviations that are used in the Replication Server documentation or that you may encounter in Replication Server messages. You can find definitions for many terms in the glossary.

Table A-1: List of acronyms

Acronym	Stands for
APC	Asynchronous Procedure Call
API	Application Program Interface
BM	Bitmap
C/SI	Client/Server Interfaces
CM	Connection Manager
dAIO	Asynchronous I/O Daemon
dALARM	Alarm Daemon
DBO	Database Owner
dCM	Connection Manager Daemon
DDL	Data Definition Language
DIST	Distributor
DML	Data Manipulation Language
dREC	Recovery Daemon
DSI	Data Server Interface
dSUB	Subscription Retry Daemon
ELM	Exceptions Log Manager
ERSSD	Embedded Replication Server System Database
EXC	Exception
EXEC	Executor
FSTR	Function String
HDS	Heterogeneous datatype support
HTS	Hash Table
LAN	Local Area Network
LL	Linked List
LTI	Log Transfer Interface
LTL	Log Transfer Language

Acronym	Stands for
MD	Message Delivery
MEM	Memory Management
MP	Multiprocessor
MSA	Multi-site Availability
NRM	Normalization
OQID	Origin Queue ID
PDS	Primary Data Server
PRS	Primary Replication Server
PRS	Parser
QID	Queue ID
RA	Replication Agent
RCL	Replication Command Language
RDS	Replicate Data Server
REP AGENT	RepAgent thread, the Replication Agent for Adaptive Server databases
RM	Replication Manager
RMI	Remote Method Invocation
RMP	Replication Manager plug-in
RMS	Replication Monitoring Services
RPC	Remote Procedure Call
RRS	Replicate Replication Server
RS	Replication Server
RSI	Replication Server Interface
RSP	Replicated Stored Procedure
RSA	Replication System Administrator
RSI	Replication Server Interface
RSSD	Replication Server System Database
SA	System Administrator
SP	Stored Procedure
SQM	Stable Queue Manager
SQT	Stable Queue Transaction Interface
SRE	Subscription Resolution Engine
STS	System Table Services
SUB	Subscription
TD	Transaction Delivery
TDS	Tabular Data Stream™
WAN	Wide Area Network

APPENDIX B Replication Server Design Limits

This appendix lists the maximum and minimum parameters and values for various replication system objects.

Replication Server limits

The variable *For_Life_Of* refers to the total number of objects that you can create for a Replication Server, regardless of whether or not any of them are dropped. For example, if the limit is 100,000, when you create 100,000, you cannot create any more, even if you drop some or all of them. The *For_Life_Of* count and limit remain in effect as long as the replication software remains installed. You can restart the *For_Life_Of* count by deleting the entire server from a system and then reinstalling it.

Table B-1: Replication Server limits

Type of object	**Number**
Replication definitions *For_Life_Of* Replication Server	2^{24} (16,777,216)
Users *For_Life_Of* Replication Server	2^{24} (16,777,216)
Reject log commands *For_Life_Of* Replication Server	2^{32} - 2^{29} (3,758,096,384)
Reject log transactions *For_Life_Of* Replication Server	2^{31} (2,147,483,648)
Replication Servers per ID Server	2^{24} (16,777,216)
Databases per ID Server	2^{24} (16,777,216)
Databases per Replication Server	2^{24} (16,777,216)
Partitions *For_Life_Of* Replication Server	2^{16} (65,536)
Minimum size for initial partition (to install RS)	20MB
Minimum size for additional partitions	1MB
Maximum partition size	1TB
Stable queues per Replication Server	2^{64}
Subscriptions *For_Life_Of* Replication Server	2^{31}

Type of object	Number
Connections per Replication Server:	
• Incoming (Replication Agent, DIST, RS, user)	2^{24} minus 1
• Outgoing (DSI, route)	2^{32} minus 1
Function strings *For_Life_Of* Replication Server	2^{24} (16,777,216)
Error classes *For_Life_Of* Replication Server	2^{24} (16,777,216)

Platform-specific limits

Certain limits specific to platform operating systems, such as number of file descriptors per process, may affect Replication Server operation. For specific limits, see the release bulletin for your platform.

Replication definition and subscription limits

Table B-2: Replication definition limits

Type of object	Number
Columns per replication definition	Limited to 1024
Primary key columns per replication definition	Limited to columns specified in the replication definition
Searchable columns per replication definition	Limited to columns specified in the replication definition
Subscriptions per replication definition	Unlimited
String width for subscription where clause	Limited to 255 bytes

Function string limits

Table B-3: Function string limits

Type of object	Number
Function strings per function-string class	Unlimited
Bytes per language-type function string template	64K
Bytes per language-type function string template after variable value substitution	64K minus 1 byte
Embedded variables per function string	Unlimited

Type of object	Number
User variables in function string input template	1024

Programming limits and parameters

Table B-4: Programming limits and parameters

Type of object	Number
Number of terms in subscription where clause	Unlimited
Transactions in a DSI transaction group	20
Source commands in a DSI command batch	50
Bytes for every command processed by Replication Server	16K
Action assignments per error class	2^{31} (2,1474,836,448)
Maximum message size written to stable queue	Unlimited

APPENDIX C RMS Server and Component States

This appendix provides information about Replication Monitoring Services (RMS) server and component states.

RMS monitors the servers and components in a replication environment, and provides information that helps you troubleshoot problems. You can monitor the replication environment either by actively viewing information about the state of servers and components, or by being notified when particular events occur.

The status of any server or component object consists of:

- An integer state value
- A list of strings that describe the reason for the current state

For example, a Replication Server can be in “Suspect” state because two different connections are “Suspended.”

The integer state value is different for each monitored object, and the descriptions can be localized.

Server states

RMS monitors the following servers:

- Replication Server
- Adaptive Server Enterprise
- Adaptive Server Anywhere and IQ
- DirectConnect
- Open Server
- Replication Agent
- RMS

Table C-1 provides a summary of server states. Details are in the sections that follow.

Table C-1: Summary of server states

Server type	Value	Description
Replication Server	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	4	SUSPECT
	6	HIBERNATE
	7	REBUILDING
	8	RECOVERY
	9	STANDALONE
	1	TIMEOUT
	10	QUIESCE
Adaptive Server Enterprise (ASE)	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	4	SUSPECT
	1	TIMEOUT
Adaptive Server Anywhere and IQ	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	1	TIMEOUT
DirectConnect	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	1	TIMEOUT
Replication Agent / Mirror Replication Agent (MRA)	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	1	TIMEOUT
	6	ADMIN
RMS	5	ACTIVE
	3	UNKNOWN
	4	SUSPECT
	0	DOWN
	1	TIMEOUT

Server type	Value	Description
Open Server	5	ACTIVE
	3	UNKNOWN
	0	DOWN
	1	TIMEOUT

Replication Server

The RMS determines the state of a Replication Server by:

1. Testing the connection to the Replication Server
2. Testing the connection to the server that contains the RSSD
3. Determining the health of the Replication Server
4. Determining the status of the server's connections, routes, and queues

The Replication Server can be in more then one state, but the RMS returns only one state. For example, the status of the server can be both HIBERNATE and QUIESCE.

Table C-2 describes the Replication Server states.

Table C-2: Replication Server states

State type	Value	Meaning	Description
Normal	5	ACTIVE	The Replication Server is running and actively replicating data.
	10	QUIESCE	The Replication Server is running but is not currently replicating data.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. UNKNOWN can also indicate that the server is not part of the replication environment.
	4	SUSPECT	At least one of the Replication Server connections, routes, or queues is down.
	6	HIBERNATE	The Replication Server is in hibernation mode. This state is returned by the admin health command.
	7	REBUILDING	The Replication Server is rebuilding queues. This state is returned by the admin health command.
	8	RECOVERY	The Replication Server is in standalone mode and is rebuilding queues. This state is returned by the admin health command.
	9	STANDALONE	The Replication Server is in standalone mode. This state is returned by admin health command.

State type	Value	Meaning	Description
Error	0	DOWN	The RMS cannot connect to the Replication Server or the server that contains the RSSD. The server state is also set to DOWN if the user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the Replication Server, or the server that contains the RSSD, timed out. This indicates a server that has stopped responding.

Adaptive Server Enterprise

The RMS determines the state of an Adaptive Server Enterprise by:

1 Testing the connection to the Adaptive Server

2 Determining the state of the Adaptive Server's RepAgent threads

RMS tests only the RepAgent thread of databases that participate in replication, and not all databases in Adaptive Server. Databases that are offline are not queried.

Table C-3 describes the Adaptive Server states.

Table C-3: Adaptive Server states

State type	Value	Meaning	Description
Normal	5	ACTIVE	Successfully connected to the Adaptive Server and all RepAgent threads for connections within this environment are enabled and started.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. This also indicates that the server is not part of the replication environment.
	4	SUSPECT	Set when the state of the RepAgent threads is checked. If any of the threads for the connections within this environment are disabled or stopped, then the server state is set to SUSPECT.
Error	0	DOWN	The RMS cannot connect to the Adaptive Server. The server state is also set to DOWN if the user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the Adaptive Server timed out. Indicates that a server has stopped responding.

Adaptive Server Anywhere and IQ

IQ is based on the Adaptive Server Anywhere (ASA) server, and both use TDS to participate in a replication environment. The RMS uses jConnect to connect to the server. Neither the ASA nor the IQ server contains internal RepAgent threads. The RMS tests the connection to the ASA or IQ server to determine its availability.

Table C-4 describes the Adaptive Server Anywhere states.

Table C-4: ASA and IQ server states

State type	Value	Meaning	Description
Normal	5	ACTIVE	Successfully connected to the ASA or IQ server.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. Also indicates that the server is not part of the replication environment.
Error	0	DOWN	The RMS cannot connect to the ASA or IQ server. The server state is also set to DOWN if the user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the ASA or IQ server timed out. Indicates that a server has stopped responding.

DirectConnect

The RMS determines the state of DirectConnect by:

1. Testing the connection to DirectConnect
2. Testing the connection from DirectConnect to the back-end data server

Table C-5 describes the DirectConnect server states.

Table C-5: DirectConnect server states

State type	Value	Meaning	Description
Normal	5	ACTIVE	RMS successfully connected to the DirectConnect, and DirectConnect can connect to the back-end data server.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. Also indicates that the agent is not part of the replication environment.
Error	0	DOWN	The RMS cannot connect to the DirectConnect. The server state is also set to DOWN if the user or password is incorrect. Additionally, the state is set to DOWN if the DirectConnect cannot connect to the back-end data server.
	1	TIMED OUT	The attempt to connect to the DirectConnect timed out. Indicates a server that has stopped responding.

Open Server

RMS tests the connection to the Open Server. Table C-6 describes the Open Server states.

Table C-6: Open Server states

State type	Value	Meaning	Description
Normal	5	ACTIVE	Successfully connected to the Open Server.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. Also indicates that the server is not part of the replication environment.
Error	0	DOWN	The RMS is unable to connect to the Open Server. DOWN can also indicate that a user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the Open Server timed out. This indicates that the server has stopped responding.

Replication Agent

The RMS determines the state of a Replication Agent by:

1. Testing the connection to the Replication Agent
2. Determining if the agent is in “administration” or “replicating” mode

Table C-7 describes the Replication Agent—including MRA and MRO—states.

Table C-7: Replication Agent (MRA/MRO) states

State type	Value	Meaning	Description
Normal	5	ACTIVE	Successfully connected to the Replication Agent. The agent is in the replicating state. This state is returned by the ra_status command.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. Also indicates that the agent is not part of the replication environment.
	6	ADMIN	Successfully connected to the Replication Agent. The agent is in the administration state and is not currently replicating data. This state is returned by the ra_status command.
Error	0	DOWN	The RMS cannot connect to the Replication Agent. The agent state is also set to DOWN if the user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the Replication Agent timed out. Indicates that an agent has stopped responding.

RMS

Central RMS tests the connection to the Remote RMS. Table C-8 describes the RMS states.

Table C-8: RMS states

State type	Value	Meaning	Description
Normal	5	ACTIVE	Central RMS successfully connected to the Remote RMS.
Warning	3	UNKNOWN	The initial value before the actual state has been determined. Also indicates that the Remote RMS is not part of the replication environment.
	4	SUSPECT	Indicates that a Remote RMS server or component is DOWN or SUSPENDED.
Error	0	DOWN	The Central RMS is unable to connect to the Remote RMS. DOWN can also indicate that a user or password is incorrect.
	1	TIMED OUT	The attempt to connect to the Remote RMS timed out. This indicates that the server has stopped responding.

Component states

RMS monitors the following components in a Replication Server:

- Connections
- Logical Connections
- Queues
- Routes
- Partitions
- RepAgent threads

Table C-9 provides a summary of component states. Details are in the sections that follow.

Table C-9: Summary of component states

Component type	Value	Description
Connection	5	ACTIVE
	2	SUSPENDED
	3	UNKNOWN
Logical Connection	5	ACTIVE
	2	SUSPENDED
	3	UNKNOWN
Queue	5	ACTIVE
	2	SUSPENDED
	6	LOSS_DETECTED
Route	5	ACTIVE
	2	SUSPENDED
	3	UNKNOWN
Partition	6	ONLINE
	7	OFFLINE
	8	DROPPED
RepAgent threads (ASE)	6	DISABLED
	7	SUSPENDED
	8	ACTIVE

Connections

The RMS monitors the state of a Replication Server's database connections. Database connections include two parts, the RepAgent and the DSI. The state of the Replication Server threads determines the state of the connection. The RMS executes the admin who command to retrieve the state of the threads.

The RMS returns the state of the DSI and RepAgent separately. Client applications such as the Replication Manager Java plug-in may consolidate the state of the threads (and the state of the actual RepAgent) when displaying the status of the connection. Table C-10 describes the connection states.

Table C-10: Connection states

State type	Value	Meaning	Description
Normal	5	ACTIVE	The Replication Server DSI or RepAgent thread is not DOWN and not SUSPENDED.
Error	2	SUSPENDED	The Replication Server DSI or RepAgent thread is DOWN or SUSPENDED.

State type	Value	Meaning	Description
Warning	3	UNKNOWN	The RepAgent for a primary connection is not part of the replication environment.

Logical Connections

The RMS monitors the state of a Replication Server's logical connections. A A logical connection consists of a pair of physical connections that are configured in a warm-standby environment. The source of the replication data is the active database while the target of replication is the standby database. Monitoring a logical connection requires the RMS to determine the state of the Replication Agent thread for the active connection and the state of the DSI for the standby connection.

RMS reports the status of the active connection's Replication Agent thread separately from the state of the standby connection's DSI thread. Each thread is reported in a separate row in the result set. Table C-11 describes the logical connection states.

Table C-11: Logical Connection states

State type	Value	Meaning	Description
Normal	5	ACTIVE	The Replication Agent for the active physical connection and the DSI thread for the standby physical connection are both active.
Error	2	SUSPENDED	The logical connection can be suspended for the following reasons: • The active or standby physical connection is not defined for the logical connection. • The Replication Agent thread for the active connection is suspended. • The DSI thread for the standby connection is suspended. • The logical connection is in the process of switching the active and standby databases.
Warning	3	UNKNOWN	The Replication Agent thread for the active connection is unknown, or the DSI thread for the standby connection is unknown.

Queues

The RMS monitors the state of Replication Server queues. Queue states are stored in the RSSD. The stored procedure rma_queue returns the name of the queue, whether the queue is up or down, and if any data loss is detected. Table C-12 describes the queue states.

Table C-12: Queue states

State type	Value	Meaning	Description
Normal	5	ACTIVE (UP)	The queue is not suspended.
Error	2	SUSPENDED	The queue is suspended.
Warning	6	LOSS_DETECTED	Data loss has been detected in the queue. The state is set to LOSS DETECTED only if the queue is UP.

Routes

The RMS monitors the state of Replication Server routes, and determines the state of a route by:

1. Checking the state of the route at both its origin and destination
2. Querying the RSSD

The RMS uses the information to identify whether the route is UP or DOWN, and to identify the reason. Table C-13 describes the route states.

Table C-13: Route states

State type	Value	Meaning	Description
Normal	5	ACTIVE	The route is open and data can pass from the origin to the destination Replication Server.
Error	2	SUSPENDED	The route is unavailable and data cannot pass between the Replication Servers. The description provides the reason for the suspended route; for example: • The route encountered an internal error. • The route is being created. • The route is suspended. • The route encountered an error at the destination. • The route is being dropped. • The route is being dropped with NOWAIT. • An indirect route is being changed to a direct route.
Warning	3	UNKNOWN	The destination Replication Server is not part of the replication environment.

Partitions

The RMS monitors Replication Server partitions. The Replication Server command admin disk_space returns the state of a partition. Table C-14 describes the partition states.

Table C-14: Partition states

State type	Value	Meaning	Description
Normal	6	ONLINE	The partition device is available and functioning normally.
Error	7	OFFLINE	The device cannot be found.
	8	DROPPED	The device has been dropped, but some queues are still using it.

RepAgent threads

The RMS monitors Adaptive Server Enterprise RepAgent threads. sp_help_rep_agent determines the state of RepAgent threads for each database that participates in replication. Table C-15 describes the RepAgent thread states.

Table C-15: RepAgent thread states

State type	Value	Meaning	Description
Normal	8	ACTIVE	The RepAgent thread is enabled and started.
Error	6	DISABLED	The RepAgent thread is not enabled.
	7	SUSPENDED	The RepAgent thread is enabled but stopped.

APPENDIX D Event Trigger Arguments

This appendix provides information about Replication Monitoring Services (RMS) event trigger arguments. Event trigger arguments contain information about the execution of a certain event, such as event name, date and time the event occurred, and name of the RMS that executed the event script. RMS passes these arguments whenever an event trigger is executed.

Connection status event arguments

Table D-1 describes the arguments of a connection status event. There are two types of connections—inbound and outbound. An inbound connection is a connection to a Replication Server from a database via a Replication Agent. An outbound connection is a connection from a Replication Server to a database.

Table D-1: Connection status event trigger arguments

Argument	Description
connection	Keyword identifying the event as a connection status event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The server where the event occurred.
source_type	The type of server that raised the event. Values are: • repserver • database
source_name	The name of the Replication Server or data server that raised the event.
ra_type	Type of Replication Agent. Values are: • rep agent • rep agent thread • dbltm • Empty string ('') if connection is outbound.
ra_name	Replication Agent name. Empty string ('') if connection is outbound.
dest_type	The destination server type. Values are: • repserver • database
dest_name	Destination server name.
state	The new connection status.

Partition status event arguments

Table D-2 describes the arguments of a partition status event.

Table D-2: Partition status event trigger arguments

Argument	Description
partition	The keyword that identifies the event as a partition status event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The name of the Replication Server that owns the partition.
part_name	The logical name of the stable device.
state	The new partition status.

Route status event arguments

Table D-3 describes the arguments of a route status event.

Table D-3: Route status event trigger arguments

Argument	Description
route	The keyword that identifies the event as a route status event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The server where the event occurred.
repserver	The keyword that identifies the origin server of the route as a Replication Server.
server_name	The name of the origin Replication Server.
thru_type	The type of intermediate server. Values are: • repserver • Empty string ('') if there is no intermediate server for the route.
thru_name	The name of the intermediate Replication Server. Empty string ('') if there is no intermediate server for the route.
repserver	The keyword that identifies the destination server of the route as a Replication Server.
dest_name	The name of the destination Replication Server.
state	The new route status.

Server status event arguments

Table D-4 describes the arguments of a server status event.

Table D-4: Server status event trigger arguments

Argument	Description
server	The keyword used to identify an event as a server status event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The server where the event occurred.
old_state	The server status before the event occurred.
new_state	The server status after the event occurred.
reason	The reason the event occurred.

Database connection latency event arguments

Table D-5 describes the arguments of a database connection latency event.

Table D-5: Database connection latency event arguments

Argument	Description
latency	The keyword that identifies the event as a latency event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The name of the Replication Server for which you are monitoring latency.
origin_dbname	Name of data server and database from which the transaction was sent.
dest_dbname	Name of the data server and database to which the transaction was sent.
delta_diff	The difference, in seconds, between the time the transaction was committed at the primary database and the time it was committed at the replicate database.
last_commit_time	The date and time of the last commit at the destination database. Format: Month Day Year HH:MM:SS:TTTMeridian
secs_since_last_commit	The time elapsed, in seconds, since the last commit.
dest_type	The type of database connection. Values are: • Primary and Replicate • Replicate Only
reason	The reason the event occurred.

Queue latency event arguments

Table D-6 describes the arguments of a queue latency event.

Table D-6: Queue latency event arguments

Argument	Description
queue_latency	The keyword that identifies the event as a queue latency status event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The name of the Replication Server that owns the queue.
log_name	The logical name of the queue.
phys_name	They physical name of the queue.
latency_in_secs	The time, in seconds, the first block has remained in a queue.

Partition and queue size threshold event arguments

Table D-7 describes the arguments of a partition and a queue size threshold event.

Table D-7: Partition and queue size threshold event arguments

Argument	Description
threshold	The keyword that identifies the event as a partition threshold or queue threshold event.
date_time	The date and time the event occurred. Format: Month Day Year HH:MM:SS:TTTMeridian
rms	The name of the RMS that executed the event script.
object_id	The name of the Replication Server that owns the partition or queue.
log_name	The logical name of the partition or queue.
phys_name	The physical name of the partition or queue.
size	Indicates the area, in percentage, used by the partition or the size, in megabytes, of the queue.
object_type	Identifies the threshold event type. Values are: • Partition • Queue

Index