



Administration Guide Volume 1

Replication Server[®] 15.7.1

SP100

DOCUMENT ID: DC32511-01-1571100-01

LAST REVISED: May 2013

Copyright © 2013 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

Conventions	1
Introduction to Replication Server	5
About Replication Server	5
Asynchronous Transaction Replication	6
Advantages of Replicating Local Data	6
Replication Server and Distributed Database Systems	7
Replication Server Basic Primary Copy Model	8
Other Distributed Data Models	11
Replication Server and Non-ASE Data Servers	15
Warm Standby Applications	16
Mixed-Version Replication Systems	17
Restrictions in Mixed-Version Systems	17
Mixed Versions of Adaptive Server	17
Replication System Security	18
Replication Server Security Features	18
Network-based Security Features	19
Replication Server Roles and Responsibilities	19
Replication System Administrator	19
Database Administrator	20
Replication Server Tasks and Responsibilities	20
Replication Server Technical Overview	23
Replication System Components	23
Replication Server	24
Replication Server System Database (RSSD)	26
Adaptive Server or Other Data Server	27
Replication Agent	28
ExpressConnect for HANA DB	29
ExpressConnect for Oracle	29
Enterprise Connect Data Access	29

Client Applications	29
System Management Tools	30
Specify Data for Replication	31
Replication Definitions and Subscriptions for Tables	32
Replication Definitions for Database Objects	32
Replication Definitions for Stored Procedures	33
Publications	34
Overview of Replicating Tables	35
Commands for Managing Replicated Data	36
Establish Replication Server Connections	36
Interfaces File	36
Kerberos	37
LDAP Server	40
Make Replication Server Connections	46
Specify Database Operations	48
Function Strings	48
Function-string Classes	49
Transaction Handling with Replication Server	49
Stable Queues	50
Distributed Concurrency Control	56
Transaction Processing by the Replication Agent	57
Manage Replication Environment with Sybase Central	61
Start and Stop Sybase Central	61
Starting Sybase Central	61
Stopping Sybase Central	62
Online Help	62
Replication Manager User Interface	63
Use Replication Manager	63
Set Up a Replication Environment	69
Preparing for a Two-tier Solution	70
Create an Environment	70

Connect to and Disconnect from a Replication Environment	71
Setting Up a Replication Environment Using Replication Manager	72
Manage Replication Server Objects	75
Monitor a Replication Environment Using RMS	79
Prepare for a Three-tier Solution	79
Connecting to RMS	79
Add and Drop Servers Through RMS	80
View Managed Objects	80
Add Event Triggers	80
Manage a Replication System	83
Set Up a Replication System	83
Create Connections and Routes	83
Set Permissions and Security	84
Verify the Replication System	84
Create Replication Definitions	84
Create Subscriptions	85
Perform Replication Server Tasks	85
Use rs_init	85
Manage Replication Server with Sybase Central	86
Use isql	86
Starting Replication Server	88
Replication Server Executable Program	88
Replication Server Configuration File	89
Stopping a Replication Server Using isql	89
Add a Replication Server	89
Add a Replication System Domain	90
Guidelines for Adding Replication System Domains	91
Set Replication Server Configuration Parameters	92
Replication Server Configuration Parameters	92
Change Replication Server Parameters	95
Manage the RSSD	96

- Enable Failover Support for an RSSD
 - Connection97
- Manage the Embedded Replication Server System
 - Database98
 - Obtain Information on ERSSD Settings98
 - ERSSD Configuration Parameters and Command99
 - Backup ERSSD100
 - ERSSD Routing100
 - Move ERSSD Files101
 - ERSSD User Administration101
 - Reduce ERSSD File Size101
 - ERSSD Recovery Procedures102
 - Quiesce Replication Server105
 - Quiescing a Replication System105
 - Remove a Replication Server106
 - Removing an Active Replication Server106
 - Removing an Inactive Replication Server108
- Manage RepAgent and Support Adaptive Server111**
 - Set up RepAgent111
 - Configuring RepAgent Using Command Line Options112
 - Configuring RepAgent113
 - Master Key and rs password114
 - Starting RepAgent115
 - Stopping RepAgent115
 - Disable RepAgent116
 - Configure RepAgent for Network Security117
 - Manage Log Transfer Activity117
 - Suspend Log Transfer117
 - Resume Log Transfer118
 - Use alter connection and the set log transfer Option119
 - Review RepAgent Status and Configuration Information119

View RepAgent Information	119
View RepAgent Configuration Parameter Values	120
View RepAgent Thread Information	120
Check Log Files for RepAgent Information and Error Messages	121
Use Counters to Monitor RepAgent Performance	121
Invoke sp_sysmon	122
Sample Output from sp_sysmon of RepAgent Activity	122
Description of Sample Output of RepAgent Counter Activity	124
Support for Extended Limits	126
Support for Longer Identifiers	127
Support for bigdatetime and bigtime Datatypes	128
System Table Support for bigdatetime and bigtime	129
Mixed-Version Information for bigdatetime and bigtime	130
Adaptive Server Shared-disk Cluster Support	130
Deferred Name Resolution	131
Adaptive Server Data Compression	132
Replication Server Support for Compressed Data	132
Maintaining Data Integrity	136
Support for Incremental Data Transfer	136
Predicated Privileges	137
Granular Permissions	138
Granular Permissions and Replication Server Configuration	138
Granular Permissions and Adding a User Database	142
Granular Permissions and User Database Upgrade	142

Granular Permissions and Master Database Replication	143
Granular Permissions and DDL Replication	143
Stored Procedure Replication and the execute as Clause	143
Precomputed Result Sets	145
Enabling Precomputed Result Sets Replication for Earlier Versions of Replication Server	146
Transfer of Database Object Ownership	147
Replication Server Support for Transferring Object Ownership	147
Backlinking Pointers and Shrinking Databases	152
Performance Enhancements for Adaptive Server Replication	152
In-memory and Relaxed-Durability Databases	153
In-Memory Databases as Primary Databases for Replication	154
In-Memory Databases as Replicate Databases for Replication	156
Restoration of In-Memory and Relaxed-Durability Databases	157
Enable Autocorrection	161
Minimal DML Logging and Replication	161
Replication Server Unsupported Operations	162
Manage Routes	165
Routing Preparation	165
Routing Rules	166
Routing Schemes	167
Direct Routes	167
Indirect Routes	167
Dedicated Routes	170
Unsupported Routing Schemes	170
Create Routes	170
create route Command	171

Configuring a Replication Server to Manage Primary Tables	175
Suspend and Resume Routes	176
Change Routes	177
Change Route Topolgy	178
Changing the Password for the RSI User for a Direct Route	180
Changing Parameters Affecting Direct Routes . .	180
Changing Configuration Parameters for All Routes	181
Routing Modification Example	182
Drop Routes	184
drop route Command	184
sysadmin purge_route_at_replicate Command .	185
Upgrade Routes	185
Monitor Routes	186
Display RSI Thread Status Using admin who ...	186
Use rs_helproute Stored Procedure	186
Manage Database Connections	189
Prepare Databases for Replication	189
Preparation of Adaptive Server Databases for Replication	189
Preparation of Non-ASE Servers for Replication	190
Upgrade an Existing Adaptive Server Database	191
Manage the Maintenance User	192
Find the Current Maintenance User	192
Grant Permissions in the Database	192
Create Database Connections	194
Information Required to Add a Database Connection	194
Use the create connection Command	195
Alter Database Connections	197
Suspend Database Connections	198

Set and Change Parameters Affecting Physical Connections	198
Resume Database Connections	224
Changing Replicate Databases to Primary Databases	224
Changing Primary Databases to Replicate Databases	226
Drop Database Connections	227
Drop a Database from the ID Server	228
Monitor Database Connections	228
View Current Database Connections	228
List Databases Managed by a Replication Server	229
Display DSI Thread Status	229
Manage Replication Server Security	231
Manage Replication Server System Security	231
Replication System Login Names	232
RSSD Login Names and Passwords	232
Replication Server Login Name and Password for RepAgent	233
ID Server Login Name and Password	234
Replication Server Login Name and Password for Replication Servers	234
Maintenance User Adaptive Server Login Name and Password	235
Maintenance User Security	235
Password Encryption	239
Send Encrypted Passwords for Replication Server Client Connections	240
Existing Encrypted Password Migration	241
Extended Password Encryption Support	241
Sybase Central Dependencies	242
Replication Server Object Creation Dependencies	242
Manage Replication Server User Security	243

Manage Replication Server Login Names and Passwords	243
Manage Replication Server Permissions	251
Examine Users, Passwords, and Permissions ..	257
Replication Server Gateway	259
Cascading Connection	259
Enable the Replication Server Gateway	259
Track Connections	260
Drop Connections	260
Limitations to Replication Server Gateway	260
Manage Network-based Security	261
How Security Services Work	261
Requirements and Restrictions	262
Set up Network-based Security	263
Maintain Network Security	292
Manage SSL Security	296
SSL Overview	296
SSL on Replication Server	297
Setting Up SSL Security on Replication Server ..	298
Enable SSL Security on Replication Server	298
Command Auditing	299
Types of Actions Recorded by Command	
Auditing	299
Classes of Commands Recorded by Command	
Auditing	299
Command Auditing Configuration	299
Security Recommendations	301
Manage Replicated Tables	303
Introduction to Managing Replicated Tables	303
Plan a Replication System	304
Design Considerations	304
Restrictions on Data Replication	305
Preparing a Replication System	306
Summary of Procedures for Replicating Tables	306

Commands for Managing Table Replication	
Definitions	309
Create Replication Definitions	310
Replication Definition Settings	310
Use the create replication definition Command	
.....	315
Create Replication Definitions Using Extended	
Limits	325
Create Multiple Replication Definitions Per Table	
.....	327
Replication Definitions and Function Strings	328
Replication Definition Restrictions in Mixed-	
Version Systems	329
Mark Tables for Replication	329
Use the sp_setreptable System Procedure	329
Replicate Java Columns	332
Restrictions to Replicating Java Columns	332
Upgrade Considerations	332
Java Datatypes in Replication Server	333
Create Replication Definitions for Java Columns	
.....	333
Function Strings for Java Columns	334
Replicate text, unitext, image, and rawobject Columns	
.....	336
Replicating Large Objects to Non-ASE Servers	
Using Enterprise Connect Direct Access	337
Guidelines for Creating a text, unitext, image, or	
rawobject Replication Definition	337
Mark Tables with text, unitext, image, or	
rawobject Columns	338
Change Column Status for text, unitext, image,	
or rawobject Columns	339
Alter Replication Status for text, unitext, image,	
or rawobject Columns	341

Subscription Issues for replicate_if_changed	
Status	342
Function Strings for Replicating text, unitext, and image Data	342
Replicate Microsoft SQL Server Large Value Data	
Types	342
Limitations of Microsoft SQL Server Large Value Data Types	343
Partial Update of Microsoft SQL Server Large Value Data Types	343
Replicate Computed Columns	343
Replicate Encrypted Columns	344
Replicate Encrypted Data	345
Work with Special Datatypes	346
Using the rs_address Datatype	346
Replicate identity Columns	347
Replicate timestamp Columns	347
Modify Replication Definitions	348
Maintain Table Schema	349
View Replication Definitions and Replication Definition Versions	349
Alter Replication Definitions	349
Drop Replication Definitions	362
Modify Replicated Data	362
Add a New Table	362
Renaming Replicated Tables	363
Add and Delete Columns in Source and Destination Tables	363
Change Searchable Columns	363
Change Column Datatypes in a Source or Destination Table	364
Use Publications	364
Use Publications to Replicate Data at the Command Line	365
Translate Datatypes Using HDS	372

Overview of Heterogeneous Datatype Support	373
Getting Started with Translating Datatypes	374
Use Class-Level and Column-Level Translations Together	377
Verify Translations	377
Manage Replicated Functions	379
Prerequisites and Restrictions for Replicated Functions	380
Replicated Function Prerequisites	380
Replicated Function Restrictions	380
Commands for Managing Function Replication	
Definitions	382
Use Replicated Functions	384
Applied Functions	384
Request Functions	387
Mark Stored Procedures for Replication	391
Subscribe to Replicated Functions	392
Modify or Drop Replicated Functions	392
Alter a Function Replication Definition	392
Modify a Function Replication Definition	393
Drop a Function Replication Definition	394
Create or Modify a Function String for a Replicated Function	394
Use Publications for Stored Procedures	395
Manage Subscriptions	397
Subscription Materialization Methods	398
Atomic Materialization	399
Nonatomic Materialization	401
Direct Load Materialization	402
No Materialization	403
Bulk Materialization	404
Bulk Copy-in and Subscription Materialization	411
Dematerialization Processing	411

Dematerialization and Purging Rows	412
Dematerialization Without Purging Rows	413
Monitor Materialization and Dematerialization	413
Verifying that the Replication System Is Ready Before You Create Subscriptions	414
Subscription Commands	416
Use the where Clause	418
Enable Replication of truncate table	419
create subscription Command	420
define subscription Command	423
activate subscription Command	424
validate subscription Command	424
check subscription Command	425
drop subscription Command	426
Subscription Example	428
Replicating Tables in the Example Replication System	428
Materialize text, unitext, image, and rawobject Data ...	431
Nonatomic Materialization	431
Row Migration	431
Subscriptions for Columns with Heterogeneous Datatypes	432
Bitmap Subscriptions	433
Obtain Subscription Information	435
Display Subscription Information	435
Verify Subscription Consistency	435
Publication Subscriptions	438
Commands for Creating and Managing Publication Subscriptions	439
Create Publication Subscriptions	441
Drop Subscriptions for Publications and Articles	445
View Publication Subscription Information	445
Manage Replicated Objects Using Multisite Availability	449

Setting Up Bidirectional Replication Support for DDL in MSA	451
Set Up an MSA System	451
Replicating the Database in a Simple Scenario	452
Replicating Tables and Functions	453
Using Replicate Databases as Warm Standby Databases	454
Mark Data for Replication	456
Manage Database Replication Definitions	457
Altering Database Replication Definitions	458
Drop Database Replication Definitions	459
Database Replication Filters	459
View Information About Database Replication Definitions	460
Concurrently Using Database, Table, and Function Replication Definitions	460
Alter Database Replication Definitions	461
Reduce the Use of Replication Definitions and Subscriptions	462
Primary Key Columns and Quoted Table or Column Names	462
Target-Scope Customized Function Strings	463
Configuring the Replication System to Reduce Replication Definitions	467
Manage Database Subscriptions	468
Materialization	469
Alter Database Subscriptions	470
Drop Database Subscriptions	470
View Information About Database Subscriptions	471
Concurrently Using Database, Table, and Function Subscriptions	471
Create and Drop Subscriptions	472
Replicating the Master Database in an MSA Environment	472

Replicating DDL and System Procedures	473
Replicate User Stored Procedures	474
Customize Function Strings	475
Manage Replication Schedules	477
Schedule Replication Tasks	477
Create Schedules	477
Alter Schedules	479
Drop Schedules	479
Display Schedules	480
Delay Replication	480
Display Delayed Replication Status	481
Index	483

Contents

Conventions

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

Style conventions

Key	Definition
monospaced (fixed-width)	<ul style="list-style-type: none"> • SQL and program code • Commands to be entered exactly as shown • File names • Directory names
<i>italic monospaced</i>	In SQL or program code snippets, placeholders for user-specified values (see example below).
<i>italic</i>	<ul style="list-style-type: none"> • File and variable names • Cross-references to other topics or documents • In text, placeholders for user-specified values (see example below) • Glossary terms in text
bold san serif	<ul style="list-style-type: none"> • Command, function, stored procedure, utility, class, and method names • Glossary entries (in the Glossary) • Menu option paths • In numbered task or procedure steps, user-interface (UI) elements that you click, such as buttons, check boxes, icons, and so on

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

Run:

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

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

Syntax conventions

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

Case-sensitivity

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

Terminology

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

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

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

Introduction to Replication Server

Replication Server® replicates data in a distributed database system. Learn the benefits and features of Replication Server, methods and concepts for replicating data, as well as defining user roles in maintaining a replication system.

About Replication Server

Replication Server maintains replicated data in multiple databases while ensuring the integrity and consistency of the data. It provides clients using databases in the replication system with local data access, thereby reducing load on the network and centralized computer systems.

Replication Command Language

The Replication Command Language (RCL) enables you to customize replication functions and to monitor and maintain the replication system. For example, you can request subsets of data for replication at the table, data row, or column level. This feature further reduces overhead by allowing you to replicate only the data that is needed at the replicate site.

Heterogeneous Data Servers

Replication Server supports heterogeneous data servers. You can build a replication system from existing databases and applications without having to convert them. As your enterprise grows and changes, you can add data servers to your replication system to meet your needs.

Replication Model

Replication Server uses a basic publish-and-subscribe model for replicating data across networks. Users “publish” data that is available in a primary database, and other users “subscribe” to the data for delivery in a replicate database. Users can replicate both changes to the data (update/insert/delete operations) and stored procedures using this method.

Instructions to publish and subscribe to data are given at Replication Servers that control, or have a connection to, each database. The user creates a replication definition at a primary Replication Server, which controls the primary database containing the data to be published. The replication definition specifies information such as which columns are to be replicated, or in the case of a database replication definition, of the database objects to be replicated. The user creates a subscription at a replicate Replication Server, which controls the replicate database that will receive the information.

Replication Server Routes

Replication Servers communicate with each other through user-defined routes. Most commonly, a primary Replication Server sends data to a replicate Replication Server through one or more routes set up to transmit data from the primary database to the replicate database.

Users may also transmit stored procedures from the replicate to the primary to request updates of the primary data; in this case, data flows through one or more routes from the replicate Replication Server to the primary Replication Server.

Connections and routes define the structure of the replication system. They allow Replication Servers to send messages to each other and to send commands to databases. A connection transfers messages from a Replication Server to a database. A route transfers requests from a source Replication Server to a destination Replication Server.

Asynchronous Transaction Replication

Replication occurs asynchronously—that is, updates to data at the primary are transferred to replicate databases in transactions separate from the update itself.

While asynchronous replication provides important advantages, system designers should remain aware of the latency between initial and replicated updates.

Advantages of Replicating Local Data

Replicating tables on local data servers provides clients with local access to enterprise data, which results in improved performance and greater data availability.

Improved Performance

In a typical Replication Server system, data requests are completed on the local data server without accessing the WAN. Local clients gain improved performance because:

- Data transfer rates are faster on a LAN than they are on a WAN.
- Local access remains unaffected by network traffic over the WAN. Local clients that share local data server resources do not compete with the enterprise-wide user community for central resources.

Greater Data Availability

Because data is replicated at local and remote databases in a Replication Server system, clients can operate in a fault-tolerant environment so that:

- When a failure occurs at a remote database, clients can use local copies of replicated data.
- When a WAN failure occurs, clients can use local copies of replicated data.
- When the local data server fails, clients can use replicated data at another site.

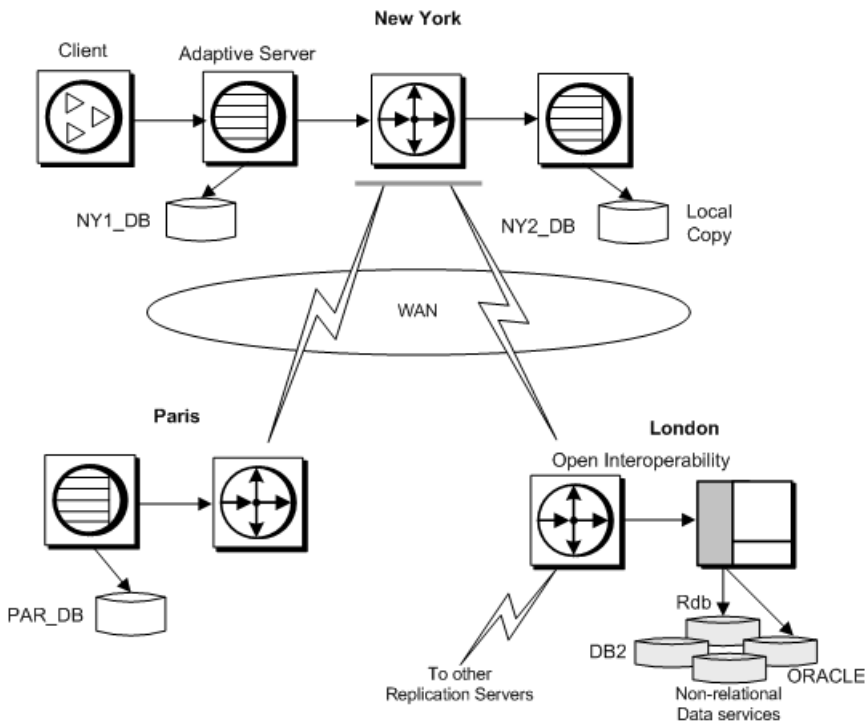
Network failure or database failure at other locations do not halt work at the local database. When WAN communications fail, Replication Server stores operations on replicated tables in *stable queues* (disk storage). The replicated tables at the unavailable databases are updated when communications resume. If a local data server fails, clients can continue working by temporarily accessing a replicate copy of the data.

Replication Server and Distributed Database Systems

Distributed database systems allow client applications to access data on multiple database servers throughout an enterprise—even geographically dispersed enterprises.

Replication Server ensures that data on replicate databases stays updated while off-loading processing responsibilities from the source database. As this figure illustrates, these enterprises may consist of many LANs and one or more WANs.

Figure 1: Replication System in a Distributed Environment



Replication Server minimizes performance and data-availability problems typically associated with remote access in distributed systems. Since Replication Server provides multiple copies of data, clients can rely on their own local data instead of remote, centralized databases. In addition, you can copy only the data you want to destination databases. Replication Server allows you to create a replication definition that identifies all or part of a table to replicate. You can then subscribe to only the rows you want. You can create a database replication definition that identifies the database objects—tables, functions, system procedures, transactions, and data definition language (DDL)—to replicate. You can also create a replication definition of a stored procedure—called a function replication definition

—to facilitate rapid replication of large amounts of data and to replicate updates from replicate databases back to the primary database. If your application requires it, you can consolidate or “roll up” replicated data from primary tables into a centralized database.

You can group replication definitions, both table replication definitions and functions replication definitions, in a publication and subscribe to them all at once. Publications allow you to organize subscriptions and then monitor them with a single command.

A Replication Agent—RepAgent for sites running Adaptive Server—transfers transaction information from a database to a Replication Server for distribution to replicate databases. Sybase also offers Replication Agent for Microsoft SQL Server, DB2, and Oracle. RepAgent is an Adaptive Server thread; all other Replication Agents are separate processes.

Several models for replicating data in distributed systems exist in Replication Server. Consult the *Replication Server Design Guide* to help you determine which model best suits your application. The model that you choose determines how you set up your system.

Setting up a replication system based on your distribution model involves:

- Creating tables to store primary and replicate data
- Setting up routes and connections between Replication Servers and establishing permissions that control access to primary data
- Creating replication definitions that identify the data you want replicated
- Creating subscriptions from replicate databases to those replication definitions

See also

- *Replication Server Technical Overview* on page 23
- *Manage a Replication System* on page 83

Replication Server Basic Primary Copy Model

The basic primary copy model is the simplest approach Replication Server uses to copy data to distribute updates from one source (primary) database to one or more destination (replicate) databases.

To ensure consistency, a source table is designated as the primary table. All other versions of the table are replicates. In this approach, replicate tables are read-only and used for operations that do not modify the data.

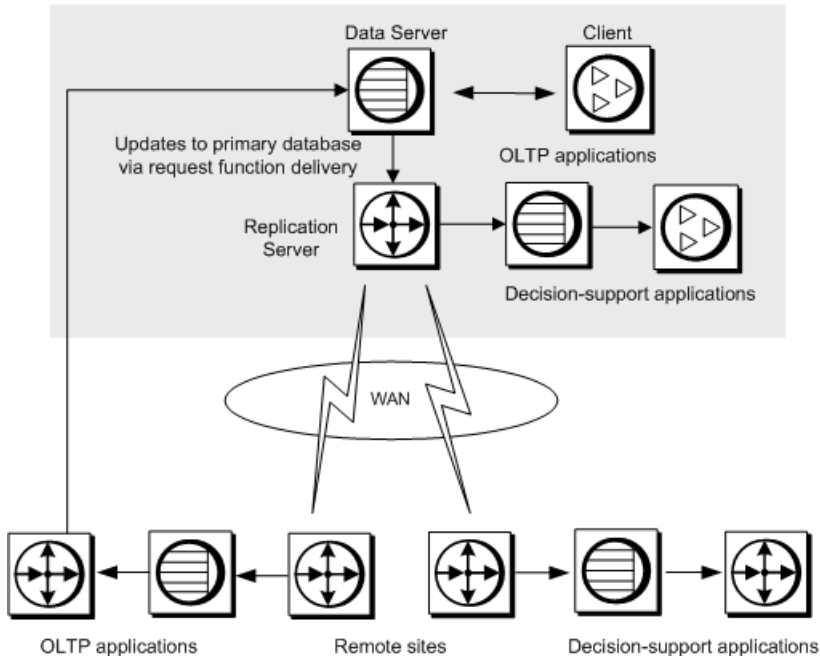
As updates occur at the primary table, Replication Server captures the updates and sends them to replicate data servers. In this model, clients at remote sites can also update primary data, either directly by accessing the primary database over the network or indirectly through replicated stored procedures.

If communication between the primary and destination databases fails, operations executed in the primary database are stored in Replication Server stable queues until they can be delivered to replicate sites. Likewise, operations executed remotely are held in stable queues until they can be delivered to the primary database.

This arrangement lets remote client applications take advantage of Replication Server fault tolerance while preserving the basic primary copy model.

This figure illustrates Replication Server configurations using the primary copy method of replicating data.

Figure 2: Replication Server Basic Primary Copy Model



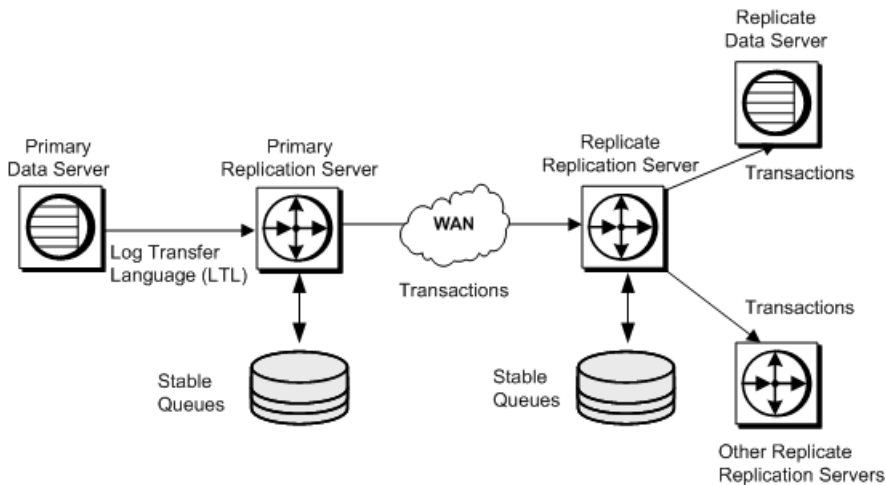
See also

- *Specify Data for Replication* on page 31
- *Manage Replicated Functions* on page 379
- *Transaction Handling with Replication Server* on page 49

Replication System Processing

A typical replication system is based on the basic primary copy model, in which a primary Replication Server and a data server are separated across a WAN from replicate Replication Servers.

Note: This example does not cover the case where primary data is updated at the replicate database.

Figure 3: Replication System Overview

This Replication System Overview diagram illustrates how data is replicated from a primary database to replicate databases. The following actions take place:

1. RepAgent reads the primary database log and converts transactions for tables or stored procedures that are marked for replication into commands that are sent to Replication Server. The Replication Server stores the transactions in a stable queue using distributed concurrency control.
2. The primary Replication Server:
 - a. Determines which Replication Servers manage replicate databases with subscriptions for the data
The primary Replication Server may have a direct route to a subscribing Replication Server or an indirect route, with one or more intermediate Replication Servers in between.
 - b. Forwards the transaction to the appropriate replicate Replication Server, where it is stored in a stable queue
 - c. Applies the transaction to any local replicate database for which there is a subscription for the data
3. The replicate Replication Server performs one or both of the following actions:
 - Routes the transaction to another Replication Server
 - Applies the transaction to replicate databases that it manages

See also

- *Distributed Concurrency Control* on page 56

Set Up a Primary Copy Model System

You need to create and configure several replication system components to set up a replication system according to the basic primary copy model.

- Set up routes and connections between Replication Servers.
- Create the table in the primary and replicate databases. The table should have the same structure in each database.
- Create indexes and grant appropriate permissions on the tables.
- Allow replication on the tables using the **sp_setreptable** system procedure.
- Create a replication definition for the table at the primary site.
- At each site, create a subscription for the table replication definition at the primary site.

See also

- *Manage Routes* on page 165
- *Manage Database Connections* on page 189
- *Manage Replication Server Security* on page 231
- *Manage Replicated Tables* on page 303
- *Manage Subscriptions* on page 397

Other Distributed Data Models

Besides the basic primary copy model, Replication Server also lets you design your system based on other distributed data models.

Other distributed data models include:

- Distributed primary fragments
- Corporate rollup
- Redistributed corporate rollup

For complete information about these distributed data models, see *Replication Server Design Guide > Implementation Strategies*.

Warm standby applications represent another type of application model. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications*.

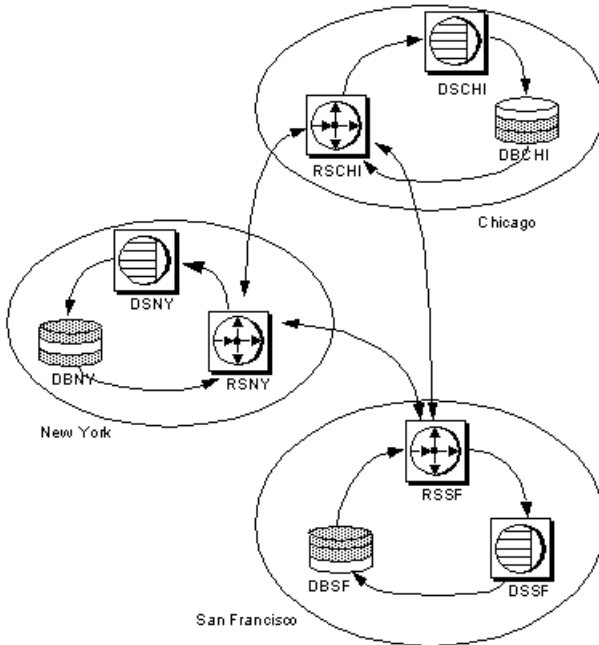
Distributed Primary Fragments

Applications that use the distributed primary fragments model include distributed tables that contain both primary and replicated data.

The Replication Server at each site distributes modifications made to local primary data to other sites and applies modifications received from other sites to the data that is replicated locally.

This figure shows the flow of data for distributed primary fragments.

Figure 4: Distributed Primary Fragments Model



The tasks needed to set up a distributed primary fragment system are similar to those for creating a basic primary copy system, with the following exceptions and additions:

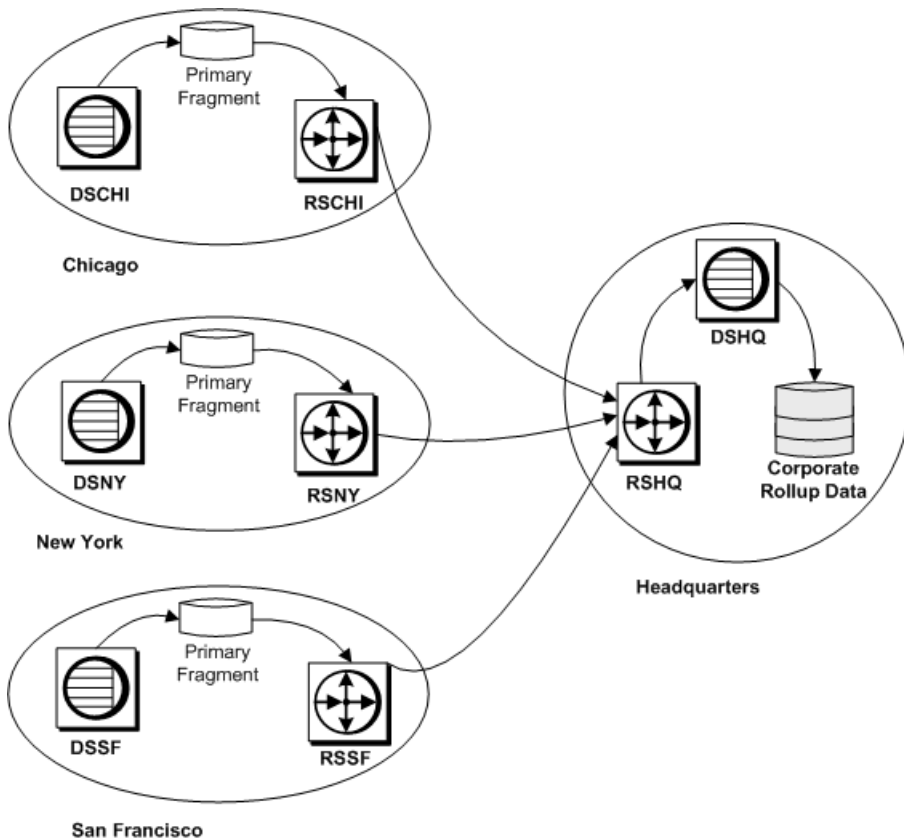
- Your application should avoid or handle cases where multiple sites update the same data at the same time. Sybase recommends that each fragment have a single “owner” site.
- Databases can be both primary and replicate. Make sure that tables with the same structure exist at both primary and replicate sites.
- Create routes from each primary site to all sites that subscribe to its data.
- Create a replication definition at any site where there is primary data, even if it is a “remote” site.
- Create subscriptions at each site for the replication definitions at the other sites. If n is the number of sites, create $n-1$ subscriptions for each replication definition.

Corporate Rollup

The corporate rollup model has distributed primary fragments and a single, centralized consolidated replicate table.

The table at each primary site contains only the data that is primary at that site. No data is replicated to these sites. The corporate rollup table is a “roll-up” of the data at the primary sites.

This figure illustrates the flow of data for a corporate rollup application model:

Figure 5: Distributed Primary Fragments with Corporate Rollup

The corporate rollup model requires distinct replication definitions at each primary site. The site where the data is consolidated subscribes to the replication definition at each primary site.

Creating a Corporate Rollup Application

You can create a corporate rollup application from distributed primary fragments.

1. Activate a Replication Agent at each primary site. However, you do not need to activate a Replication Agent at the central site, since data is not replicated from that site.
2. Create tables in each primary database and in the database at the central site.
3. Allow for replication on tables at each remote database where primary data is stored.
4. Create replication definitions for tables at each remote site where primary data is stored.
5. At the headquarters site, where the data is to be consolidated, create subscriptions for the replication definitions at the remote sites.

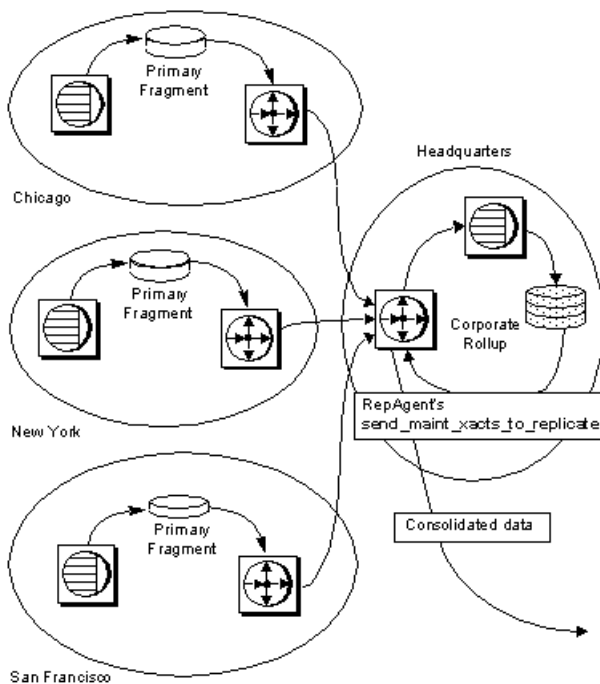
Redistributed Corporate Rollup

The redistributed corporate rollup model is similar to the corporate rollup model.

Primary fragments distributed at remote sites are rolled up into a consolidated table at a central site. At the site where the fragments are consolidated, however, a Replication Agent processes the consolidated table as if it were primary data. The data is then forwarded to Replication Server for distribution to subscribers.

This figure illustrates the flow of data in an application based on the redistributed corporate rollup model:

Figure 6: Distributed Fragments with Redistributed Corporate Rollup



The consolidated table is described with a replication definition. Other sites can then subscribe to this table. Do not allow applications to update the corporate rollup table directly. All updates should originate from the primary sites.

The tasks associated with creating a redistributed corporate rollup replication system are identical to the corporate rollup model, except that:

- A Replication Agent must be activated at the headquarters site for the consolidated database so that all updates are submitted to the Replication Server as if they were made by a client application.
RepAgent must be configured with its **send_maint_xacts_to_replicate** option set to **true**. Otherwise, the Replication Agent filters will not redistribute replicated data as primary data.
- A Replication Agent is required for the headquarters Replication Server, since data will be redistributed from that site.
- At the headquarters site a replication definition must be created for each table. Other sites can create subscriptions to this replication definition, but the primary sites cannot subscribe to their own primary data.
- The headquarters Replication Server must have routes to the other sites that create subscriptions for the consolidated replicate table. If the primary sites create subscriptions, routes must be created to them from headquarters.
- Do not allow rollup sites to re-create subscriptions to their primary data. If they do, transactions could loop endlessly through the system.

See also

- *Manage a Replication System* on page 83

Replication Server and Non-ASE Data Servers

Replication Server supports non-ASE data servers through an open interface. You can use any data-storage system as a data server if it supports a set of required basic data operations and transaction-processing directives.

Sybase provides:

- ExpressConnect for HANA DB – to connect Replication Server with HANA databases
- ExpressConnect for Oracle – to connect Replication Server with Oracle data servers
- Sybase Enterprise Connect™ Data Access (ECDA) – to connect Replication Server with other supported non-ASE data servers

If a data server does not support Sybase Open Client™ and Open Server™, you can create an Open Server gateway to allow Replication Server to access the data server.

For detailed information about using Replication Server with databases from different vendors, see the *Replication Server Heterogeneous Replication Guide*.

Other open architecture components include:

- Replication Agents

A Replication Agent detects modifications made to primary data and submits them to Replication Server for distribution to other databases.

The RepAgent thread in Adaptive Server is the Replication Agent for Adaptive Server databases.

If you use non-Adaptive Server data servers, you must provide a replication agent for them. Replication Agents for Microsoft SQL Server, IBM DB2 UDB, and Oracle

databases are available from Sybase. For details, see the *Replication Server Design Guide* and Sybase documentation for Replication Agent.

- Error classes and error processing actions

Error classes allow you to tailor your system to handle database errors for a type of data server. You can specify error actions in response to errors that a data server returns to Replication Server. Replication Server provides a default error class for Adaptive Server. Replication Server 15.2 and later provides default error classes for HANA DB, Oracle, Microsoft SQL Server, and IBM DB2 UDB databases. See *Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions*.

- Functions, function strings, and function-string classes

Replication Server uses function strings to format replicated operations correctly for a type of destination database. To aid replication system administrators, Replication Server groups all the function strings for a particular type of database into a function-string class. Replication Server provides default function-string classes for Adaptive Server and provides connection profiles that install function-string classes for HANA DB, Oracle, Microsoft SQL Server, and IBM DB2 UDB databases. You can customize function strings to execute commands appropriate for your database and application. See *Replication Server Administration Guide Volume 2 > Customize Database Operations* for details.

Warm Standby Applications

Use Warm standby applications to maintain a set of databases, one or more of which functions as standby copies of an active database.

As clients update the active database, Replication Server copies transactions to the standby databases, maintaining consistency between them. Should the active database fail for any reason, you can switch to a standby database, making it the active database, and resume operations with little interruption.

Replication Server provides two methods for setting up a warm standby application. In both methods, the active and standby databases must be Adaptive Server or Oracle databases. They can act as either a primary or replicate database with respect to other databases in the system.

- One method uses the multisite availability (MSA) feature to set up an active and one or more standby databases.
- The second method lets you set up an active and a single standby database, both of which must be managed by the same Replication Server. This warm standby application is considered a single logical unit in a Replication Server system.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications* to set up warm standby applications for Adaptive Server databases.

See *Replication Server Heterogeneous Replication Guide > Heterogeneous Warm Standby for Oracle* to set up warm standby applications for Oracle databases.

See also

- *Manage Replicated Objects Using Multisite Availability* on page 449

Mixed-Version Replication Systems

A replication system can include Replication Servers or Adaptive Servers of different versions. Each system presents different issues.

- If a replication system domain has Replication Server 15.5 and later, then the system version, and all site and route versions in the replication system domain, must be version 12.6 and later.

You must upgrade Replication Server to version 12.6 or later, set site version to 12.6 or later, and upgrade routes to 12.6 or later, before you can upgrade to version 15.5 or later. See *Replication Server Configuration Guide > Upgrade or Downgrade Replication Server*.

- When all Replication Servers are at least version 12.6 and the system version is set to 12.6, each Replication Server uses features according to its *site version*. For example, Replication Servers running version 15.5 can use all 15.5 features among themselves, while Replication Servers running 15.0 can only use 15.0 features. Such a system is called a *mixed-version system*; each Replication Server can use all of its features.

Restrictions in Mixed-Version Systems

Interaction between Replication Servers of different versions is restricted to the capabilities of the oldest version.

Information associated with new features may not be available to Replication Servers of earlier versions. See the documentation for each feature introduced in a new version, such as function-string inheritance or multiple replication definitions, for additional information about usage restrictions in mixed-version environments.

Refer to the installation and configuration guides and the release bulletin for your platform for more information about mixed-version systems and about setting the site version and system version.

Mixed Versions of Adaptive Server

You can use Replication Server version 15.0 or later with different versions of Adaptive Server.

Although you can use data sources and destinations other than Adaptive Server, Replication Server requires Adaptive Server for warm standby databases and either Adaptive Server or SQL Anywhere® for Replication Server System Databases (RSSD).

Note: Sybase does not support replication of Adaptive Server system databases, such as `tempdb`, `model`, `sybsystemprocs`, `sybsecurity`, and `sybssystemdb`. The

replication of the Adaptive Server system database master is supported only if the Adaptive Server supports master database replication.

Some capabilities of Replication Server version 15.0.1 require you to use an Adaptive Server version 15.0.1 or later.

Some capabilities of Replication Server version 15.2 require you to use an Adaptive Server version 15.0.3 or later. For example, the SQL statement replication feature in Replication Server version 15.2 requires Adaptive Server version 15.0.3 or later.

Refer to the installation and configuration guides and the release bulletin for your platform for more information about using Adaptive Server with Replication Server.

Replication System Security

Replication Server provides careful management of the login names, passwords, and permissions that are essential for system security. In addition, Replication Server supports third-party security mechanisms that safeguard data transmission across the network.

See also

- *Manage Replication Server Security* on page 231

Replication Server Security Features

Replication Server enforces security using several features.

Replication Server enforces security using login names, encryption, and permissions.

- Replication Server login names
Each Replication Server has its own set of login names, which are distinct from data server login names. This distinction gives the replication system administrator control over replicated data and other aspects of the replication system.
- Data server login names
Data server login names are used with client applications to connect to data servers. Clients are generally given permission to update primary data. On replicate tables, however, clients are generally granted permission to select or view data, but are prohibited from making changes to data. These permissions are controlled in the data server, according to the application.
- Data server maintenance user login names
Replication Server uses a special data server *maintenance user* login name for each local data server database that contains replicate tables. This allows Replication Server to maintain and update the replicate tables in the database.
- Password encryption
You can encrypt passwords in sensitive areas of the replication system.
- Permission system

Replication Server permissions are assigned to and cancelled from Replication Server login names using the **grant** and **revoke** commands.

See also

- *Replication Server Roles and Responsibilities* on page 19

Network-based Security Features

Replication Server supports third-party, network-based security mechanisms.

Third-party network-based security mechanisms can:

- Establish unified logins to servers on the network.
The security mechanism authenticates users at login. Each authenticated user is given a security credential that can be presented to remote servers as needed. As a result, users can seamlessly access different servers using a single login.
- Ensure secure data transmission across the network
A choice of different data protection services can:
 - Encrypt and decrypt data transmissions
 - Verify that a transmission has not been tampered with
 - Verify the origin of each transmission
 - Verify that a transmission has not been captured and re-sent
 - Verify that transmissions are received in the order sent

See also

- *Manage Network-based Security* on page 261

Replication Server Roles and Responsibilities

Administering the replication system is primarily the role of the replication system administrator. The database administrator plays a subsidiary role by supporting some Replication Server administration tasks. At some sites, role distinctions may not be clear-cut and some responsibilities can overlap.

Replication System Administrator

The replication system administrator installs, configures, and administers the replication system.

On a WAN, this role may be performed by different people at different locations. If this is the case, various tasks for administering Replication Server may require coordination between replication system administrators.

The replication system administrator has **sa** user permissions, which provide that person with the ability to execute nearly all commands in the replication system. In managing the system,

the replication system administrator may need to coordinate with database administrators for both local and remote databases.

Database Administrator

The database administrator has two responsibilities.

The database administrator is responsible for:

- Administering local data servers, including login names and permissions.
- Managing data in a distributed database system. Various tasks may require coordination between database administrators for different databases.

Replication Server Tasks and Responsibilities

Use the Replication Server Tasks and Responsibilities table to assign tasks and roles to maintain the replication system.

Table 1. Replication Server Tasks and Responsibilities

Task	References	Roles
Installing Replication Server and initial configuration.	Installation and configuration guides	replication system administrator (RSA), database administrator (DBA)
Starting up and shutting down Replication Server.	<i>Manage a Replication System</i> on page 83	RSA
Quiescing Replication Server.	<i>Manage a Replication System</i> on page 83	RSA, DBA
Adding login names, database users, and administering appropriate permissions.	<i>Manage Replication Server Security</i> on page 231	RSA, DBA
Monitoring Replication Server.	<i>Manage a Replication System</i> on page 83	RSA
Configuring Replication Server.	<i>Manage a Replication System</i> on page 83	RSA

Task	References	Roles
<p>Adding replicated tables and stored procedures or changing table schemas.</p> <ul style="list-style-type: none"> • Creating and modifying replicated tables and stored procedures. • Creating and modifying table and function replication definitions. • Creating and materializing subscriptions at replicate sites. 	<ul style="list-style-type: none"> • <i>Manage Replicated Tables</i> on page 303 • <i>Manage Replicated Functions</i> on page 379 • <i>Manage Subscriptions</i> on page 397 	RSA, DBA
<p>Defining data server function-string classes and function strings.</p>	<p><i>Replication Server Administration Guide Volume 2 > Customize Database Operations</i></p>	RSA, DBA
<p>Maintaining routes.</p> <ul style="list-style-type: none"> • Creating and modifying routes. 	<p><i>Manage Routes</i> on page 165</p>	RSA
<p>Maintaining and monitoring database connections.</p> <ul style="list-style-type: none"> • Suspending and resuming connections. 	<p><i>Manage Database Connections</i> on page 189</p>	RSA
<p>Creating a warm standby application.</p>	<p><i>Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications</i></p>	RSA, DBA
<p>Localizing Replication Server.</p>	<p><i>Replication Server Design Guide</i></p>	RSA
<p>Adding a primary or replicate database connection.</p>	<p><i>Manage Database Connections</i> on page 189</p>	RSA, DBA
<p>Adding or removing a Replication Server.</p>	<p><i>Manage a Replication System</i> on page 83</p>	RSA
<p>Processing rejected transactions.</p>	<p><i>Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions.</i></p>	RSA, DBA

Task	References	Roles
Administering local data server. <ul style="list-style-type: none"> Suspending or resuming data server. 	Adaptive Server or local server documentation.	DBA
Managing the RSSD.	<i>Manage a Replication System</i> on page 83	RSA, DBA
Managing Embedded Replication Server System Database (ERSSD).	<i>Manage a Replication System</i> on page 83	RSA, DBA
Creating, deleting, and modifying databases for replication.	Adaptive Server or local server documentation.	DBA
Setting up database user login names and passwords.	Adaptive Server or local server documentation.	DBA
Performing regular backups.	<i>Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server.</i>	DBA
Applying database recovery procedures.	<i>Replication Server Administration Guide Volume 2 > Replication System Recovery</i>	RSA, DBA
Reconciling database inconsistencies.	<i>Manage Subscriptions</i> on page 397	RSA, DBA
Auditing user actions performed on Replication Server.	<i>Command Auditing</i> on page 299	RSA

Replication Server Technical Overview

Acquire a technical overview of Replication Server and the replication system.

Learn about:

- Components of a distributed database system based on Replication Server
- Movement of transactions from a primary database to a replicate database.
- Aspects of Replication Server that play a role in receiving and distributing data at primary and replicate sites.
- Diagnosing and troubleshooting replication system problems.

Replication System Components

A replication environment consists of components and resources that must be present or assembled before you can run Replication Server.

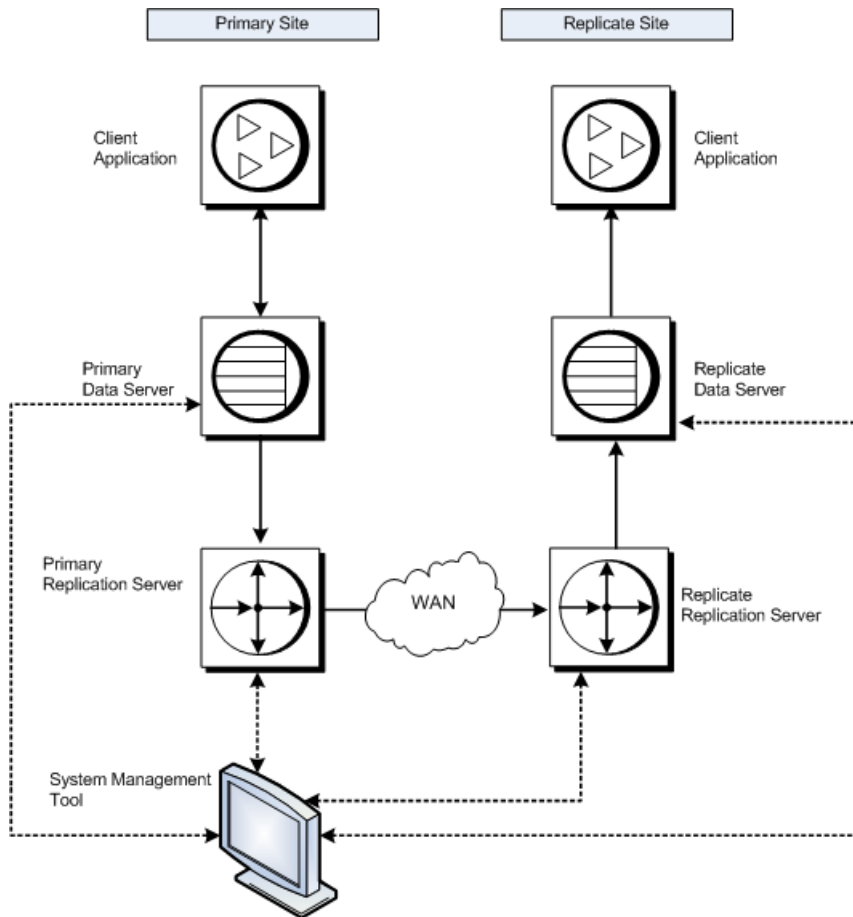
Components in a Replication Server environment can include:

- Replication Server
- Data servers such as Adaptive Server.
- Replication Server System Database (RSSD)
- Replication Agent, ExpressConnect for HANA DB, ExpressConnect for Oracle, and ECDA
- Client applications
- System management tools such as Sybase Control Center or Sybase Central™

Each component uses the Open Client/Server™ Interface to communicate with other components.

This Replication Server Domain diagram illustrates a simple configuration for a WAN-based distributed database system based on Replication Server.

Figure 7: Replication Server Domain



Replication Server

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

Replication Server provides guaranteed delivery of transactions to each replicate site by:

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

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

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

See also

- *Replication Server System Database (RSSD)* on page 26

ID Server

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

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

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

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

Because of these requirements, the ID Server is the first Replication Server that you install and start when you install a replication system. If you have only one Replication Server, or if you are installing Replication Server for the first time, then that Replication Server is also the ID Server. If you are adding a Replication Server to an existing replication system, you must know the name of the Replication Server in the system that is the ID Server.

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

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

See also

- *Manage a Replication System* on page 83

Replication System Domain

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

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

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

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

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

See also

- *Add a Replication System Domain* on page 90

Replication Server System Database (RSSD)

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

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

System Tables

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

System tables hold information such as:

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

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

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

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

To query the system tables and find status information:

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

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

RSSD and Replication Agent Specifications

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

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

The RSSD is dedicated to the Replication Server that it supports; do not use it to store user data. However, a single data server may contain the RSSD and user databases. The database device space for the RSSD must be at least 20MB (10MB for data and 10MB for the log). It is best to put the database and the database log on separate devices.

See also

- *Manage Routes* on page 165

Adaptive Server or Other Data Server

Adaptive Server or other data servers in a replication system, manage databases containing either primary or replicate data.

Client applications use Adaptive Server to store and retrieve data and to process transactions and queries.

Replication Server Technical Overview

Each Replication Server requires an Adaptive Server database for its Replication Server System Database (RSSD) or an SQL Anywhere database for its Embedded Replication Server System Database (ERSSD), which contains the Replication Server system tables.

Replication Server also supports non-ASE data servers through an open interface. You can use any system for storing data if it supports a set of required basic data operations and transaction processing directives. For data servers that contain primary databases, you must use a compatible Replication Agent program.

Actively supported data servers are data servers for which Sybase provides all the required software, documentation, and support for the data servers to serve as both a primary or a replicate data server.

See the *Replication Agent Release Bulletin* for the list of actively supported non-ASE data servers and see the *Replication Server Heterogeneous Replication Guide* for details on heterogeneous data server support.

Replication Agent

A Replication Agent notifies Replication Server of actions in a primary database that must be copied to other databases.

The Replication Agent reads the database transaction log and transfers log records for replicated tables and stored procedures to the Replication Server managing the database, which distributes the modifications to databases that subscribe to the data.

A Replication Agent is needed for every database that contains primary data and for every database where stored procedures that need to be replicated are executed. A database that contains replicated data and no stored procedures marked for replication does not require a Replication Agent.

Replication Agents communicate with Replication Server by executing commands in Log Transfer Language (LTL).

See *Replication Server Design Guide > Introduction to Replication Agents* for more information about LTL commands.

Which Replication Agent for Your System?

The Replication Agent you use depends on the data servers in the replication system.

Supported Replication Agents are:

- RepAgent – for Adaptive Server data servers. RepAgent, a thread in Adaptive Server, is the Replication Agent described in the *Replication Server Administration Guide*.
- Replication Agents available for non-ASE data servers:
 - IBM DB2
 - IBM DB2 UDB
 - Microsoft SQL Server

- Oracle

ExpressConnect for HANA DB

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

ExpressConnect for HANA DB, which is available with Replication Server 15.7.1 SP100 and later, eliminates the need for installing and setting up a separate gateway server, thereby improving performance and reducing the complexities of managing a replication system.

See the *Replication Server Heterogeneous Replication Guide* and the Replication Server Options documentation.

ExpressConnect for Oracle

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

ExpressConnect for Oracle which is available with Replication Server Options 15.5 and later, eliminates the need for installing and setting up a separate gateway server, thereby improving performance and reducing the complexities of managing a replication system.

See the *Replication Server Heterogeneous Replication Guide* and the Replication Server Options documentation.

Enterprise Connect Data Access

Sybase Enterprise Connect™ Data Access (ECDA) is an integrated set of software applications and connectivity tools that allows you to access data within a heterogeneous database environment.

ECDA gives you the ability to access a variety of LAN-based, non-Sybase data sources, as well as mainframe data sources. It consists of components that provide transparent data access within an enterprise. You require a specific ECDA component for each actively supported non-ASE database that does not already provide an Open Server interface. This is required for replication to the non-ASE database.

See the *Replication Server Heterogeneous Replication Guide* and the Replication Server Options documentation.

Client Applications

A client application is a program that accesses a data server.

When the data server is Adaptive Server, applications can be programs created with Open Client Client-Library™ or DB-Library™, Embedded SQL™, or any other front-end development tool that is compatible with the Open Client/Server Interfaces™ (C/SI) such as PowerBuilder®. Open Client/Server includes routines and protocols for client/server communications.

In a simple replication system, clients update primary databases and Replication Server updates replicate databases. By replicating stored procedures, clients can update primary data from any replicate database.

System Management Tools

You can use system management tools such as Sybase Control Center and Sybase Central™, to manage your replicaton system.

Sybase Control Center for Replication

Sybase® Control Center for Replication is a Web-based solution that replaces Replication Monitoring Services (RMS), a middle-tier server, in monitoring the status and availability of servers in a replication environment.

Sybase Control Center for Replication allows you to monitor and manage large, complex, and geographically dispersed replication environments. It lets you search, sort, and filter servers and component objects to support a larger environment than what the current Replication Manager and Replication Monitoring Services can handle.

Sybase Control Center for Replication provides status information at a glance, using server monitors and a heat chart for displaying the availability or status of a specific server. The server monitors display high-level information, such as server version and platform. The server monitors also display critical performance counters to aid you in troubleshooting replication performance.

To help you control the flow of data and configure replication parameters to improve server performance, Sybase Control Center for Replication provides a quick administration tool that you can easily access through every replication monitor.

In addition to the monitors, Sybase Control Center for Replication provides a topology view that graphically displays the servers, the connections between servers, data flow in the environment, and a replication path's sources and targets. Graphs and charts are also available for monitoring performance counters.

See *Sybase Control Center > Sybase Control Center for Replication*.

Sybase Central

Sybase Central is a graphical management tool for Sybase products.

It implements the Sybase enterprise management strategy, which calls for a single management console, seamlessly integrated, across all server and middleware products. It connects to and manages Sybase products that are running on any Sybase-supported platform. Sybase Central is available on all Windows and UNIX platforms that Replication Server supports.

The Replication Manager is a plug-in to Sybase Central, which allows you to develop, manage, and monitor a replication environment.

Replication Manager (RM) Plug-in to Sybase Central

The Replication Manager plug-in to Sybase Central is a management utility for developing, managing, and monitoring replication environments.

With its easy-to-use interface, Replication Manager allows you to perform many administrative tasks that you would otherwise use RCL commands to perform, including:

- Creating, altering, and deleting Replication Server objects such as, connections, routes, replication definitions, and subscriptions.
- Managing, monitoring, and troubleshooting replication system components.
- Monitoring the availability of servers and the state of connections and routes.
- Generating the RCL scripts for all Replication Server objects. Providing a script editor window that allows users to submit RCL or SQL to a server.
- Managing a replication domain, including configuration parameters for Replication Servers, Replication Agents, RepAgent threads, connections, and routes.
- Controlling the flow of data by suspending and resuming connections and routes.
- Displaying transactions in Replication Server stable queues.
- Displaying transactions in Replication Server exception log and allowing the user to edit and resubmit transactions.
- Managing a warm standby environment.

Replication Monitoring Services (RMS)

Replication Monitoring Services is a monitoring tool that you can use if your replication environment is fairly complex, involving ten or more servers.

RMS allows you to monitor various servers and components in your environment, acting as a middle layer between the Replication Manager and the servers in the replication environment. RMS also provides the ability to control the flow of data and set the configuration parameters.

Specify Data for Replication

Replication Server lets you define the data and stored procedures that you want to replicate at remote databases, as well as letting you specify the destination databases themselves.

Replication Server uses the relational database model to represent data in tables that have a fixed number of columns and a varying number of rows. Each table you want to replicate must have one or more columns that can be used as a primary key to uniquely identify each row.

As part of design and planning, you designate source and destination databases for your replication system and create the routes that replicated data follows from one Replication Server to another.

In general, a source database contains primary data and may be called the primary database, while a destination database contains replicate data and may be called the replicate database. Depending on your implementation, the same database may contain both primary and

replicate data. Transactions or stored procedure executions are replicated from primary to replicate databases. Stored procedure executions may also be replicated from replicate to primary databases.

See also

- *Replication Server Basic Primary Copy Model* on page 8

Replication Definitions and Subscriptions for Tables

You create one or more replication definitions to describe each primary (source) table.

A replication definition lists a table's columns and datatypes, the columns that make up the primary key, the columns that can be used in subscribing to the primary data, and specifies the location of the primary version of the table.

A replication definition may include additional settings to let you customize how you will use it. For example, you can create a replication definition just for replicating into a standby database in a warm standby application.

You then create subscriptions for transactions on the data defined in the replication definition. A subscription instructs Replication Server to copy transactions for all rows or for qualifying rows only. Copies of a table can be limited to only the rows or columns needed.

Typically, creating a subscription causes Replication Server to copy the initial requested data from the primary database to the replicate database. This process is called subscription materialization. Once the subscription is created and materialized, Replication Server begins distributing database operations for the primary data as they occur.

See also

- *Manage Replicated Tables* on page 303
- *Manage Subscriptions* on page 397

Replication Definitions for Database Objects

You can use multisite availability (MSA) to create a single database replication definition to describe the data to be sent to the replicate database.

The database replication definition describes the database objects that are to be replicated. You can choose to replicate, or not replicate, individual tables, transactions, functions, system stored procedures, and DDL.

You then create a single database subscription at each subscribing database for the data described in the database replication definition. Database subscriptions cannot limit the data copied.

MSA provides a simple replication methodology that requires only one replication definition for the primary database and only one subscription for each subscribing database. If you want to transform the data or if a table needs to have a different **replicate minimal columns** or

dynamic SQL setting than the target connection, you must add table and function replication definitions.

See also

- *Manage Replicated Objects Using Multisite Availability* on page 449

Replication Definitions for Stored Procedures

A replication definition of a stored procedure is called a function replication definition.

For certain operations, replication of stored procedures may offer significant performance improvements over table replication. In addition, you can replicate stored procedures to update data from a replicate database to a primary database.

See also

- *Deferred Name Resolution* on page 131
- *Manage Replicated Functions* on page 379
- *Manage Subscriptions* on page 397

Benefits of Replicated Functions over Normal Replication

Learn the benefits of replicated functions.

Adaptive Server logs a record for each row modified by a Transact-SQL® command. When a single Transact-SQL command modifies multiple rows, Replication Server treats each log record received from the Replication Agent as a separate command in the transaction. For example, to replicate the results of a single **update** command that modifies 1000 rows in the primary database, Replication Server may execute 1000 **update** commands in each replicate database.

Commands that modify many rows can affect performance of replicate Adaptive Servers and the replication system. The volume of rows delivered through the replication system may use all available space in stable queues.

If an application updates multiple rows in a primary table, you can use replicated stored procedures to maintain data in destination databases. Because commands in stored procedures can modify multiple rows, using stored procedures allows you to update rows in replicate databases without passing images of the rows through the replication system. Only a single record reflecting stored-procedure execution and its parameters replicates through the system.

You can use SQL statement replication to improve the performance for a single Transact-SQL command which modifies multiple rows. See *Replication Server Administration Guide Volume 2 > Performance Tuning > SQL Statement Replication*.

In Replication Server, you can enable support for the Adaptive Server deferred name resolution feature.

See also

- *Manage Replicated Functions* on page 379
- *Manage Subscriptions* on page 397
- *Deferred Name Resolution* on page 131

Use Replicated Functions

A function replication definition describes a replicated stored procedure.

A function replication definition includes:

- The parameters and datatypes
- The location of the primary data that the stored procedure may modify
- Parameters that can be used in subscribing to stored-procedure executions
- The name of the stored procedure to execute at the destination database

There are two types of replicated function delivery:

- **Applied** – executed at primary databases first and affect primary data. Replication Servers propagate the stored procedure and its parameters, applying data changes asynchronously at replicate sites that have subscriptions for an applied function replication definition. The maintenance user executes the applied function at the replicate sites.
- **Request** – executed at primary databases first and affect primary data. Replication Servers propagate the stored procedure and its parameters, applying data changes asynchronously at replicate sites that have subscriptions for a request function replication definition. The same user who executes the stored procedure at the primary databases executes the request function at the replicate sites.

Typically, the request function delivery is used to modify the remote data asynchronously at databases on other sites. The changes are replicated back to the originating site via either normal data replication or applied function delivery.

See also

- *Manage Replicated Functions* on page 379
- *Manage Subscriptions* on page 397

Publications

A publication lets you collect replication definitions for related tables and stored procedures and then subscribe to them as a group.

You create publications at the primary Replication Server and subscribe to them at the destination Replication Server.

When you use publications, you create and manage these objects:

Article – identifies a replication definition, primary database, and publication. It may also limit the number of rows or parameters sent to the replicate database.

Publication – a collection of articles from a primary database.

Publication subscription – a subscription to a publication. When you create a publication subscription, Replication Server creates a subscription for each article in the publication.

Publications allow you to group replication definitions and subscriptions in a manner that makes sense for your system. It also allows you to create and check the status of only one subscription for a set of tables and procedures.

Overview of Replicating Tables

Learn the tasks you must perform at the data servers and Replication Servers in a replication system to replicate transaction data from a primary (source) to a replicate (destination) table.

1. At the replicate data server: Create a copy of a table into which data will be replicated from the primary table. The copy may contain all or a subset of the columns from the primary table.
2. At the primary Replication Server: Create a replication definition to identify the table data you want to replicate. You can create one or more replication definitions per table that can be replicated into different replicate databases. You can also create replication definitions for stored procedures.

After you have created a replication definition, transactions are available for replication to qualifying destination Replication Servers that subscribe to the replication definition.

You can create a set of articles that reference replication definitions and group them in a publication. If you want to limit the transactions sent to the replicate database to those that affect certain rows, use a **where** clause in the article.

3. At the primary data server: Use the **sp_setreptable** system procedure to mark a table as replicated if Adaptive Server is the primary data server. If you use a different data source with a Replication Agent, refer to your Replication Agent documentation for information about marking primary objects for replication.

When you mark a table as replicated in the primary data server, the Replication Agent for the primary database can forward the table's transactions to the primary Replication Server.

If you want to replicate `text`, `unitext`, or `image` columns, you may also need to use the **sp_setrepcol** system procedure.

4. At replicate Replication Servers: Create a subscription for replication definitions that were created in primary Replication Servers. A subscription allows the replicate table to receive the initial data from the primary table through a process known as *materialization*, and to begin receiving subsequent replicated data updates.

You can create multiple subscriptions for each replication definition, but a replicate table can subscribe to only one replication definition. You can set up a subscription to receive all transactions for a replicate table, or use a **where** clause to receive just the transactions that affect certain rows.

Create publication subscriptions for publications created at the primary Replication Server. When you do so, Replication Server creates an article subscription for each article in the publication.

Creating subscriptions completes the process of replicating data.

See also

- *Mark Tables for Replication* on page 329
- *Manage Replicated Functions* on page 379
- *Manage Subscriptions* on page 397
- *Replicating Tables in the Example Replication System* on page 428

Commands for Managing Replicated Data

Replication Manager plug-in help lists tasks and concepts for working with table replication definitions, function replication definitions, and subscriptions in Sybase Central. Replication Server also provides table replication definition, function replication definition, publication, subscription, and publication subscription RCL commands to manage replicate data.

See also

- *Commands for Managing Table Replication Definitions* on page 309
- *Commands for Managing Function Replication Definitions* on page 382
- *Commands for Creating and Managing Publications* on page 365
- *Subscription Commands* on page 416
- *Commands for Creating and Managing Publication Subscriptions* on page 439

Establish Replication Server Connections

Replication Server uses the Open Client/Server Interfaces to communicate between client applications and servers.

Server programs, including Replication Servers, Adaptive Servers, and gateway software for other data servers, are registered in a directory service—either an `interfaces` file, Kerberos, or a Lightweight Directory Access Protocol (LDAP) server—so that client applications and other server programs can locate them.

Note: If you are using network-based security, use the directory services of your network security mechanism to register Replication Servers, Adaptive Servers, and gateway software. Refer to the documentation that comes with your network-based security mechanism for details.

Interfaces File

The `interfaces` file contains network definitions for servers in the replication system, including Replication Servers and data servers.

Generally, one `interfaces` file at each site contains entries for all local and remote Replication Servers and data servers. The entry for each server includes its unique name and the network

information that other servers and client programs need to connect with it. The interfaces file at a site requires entries for these components:

- ID Server (if Replication Server is not also the ID Server)
- Replication Server
- RSSD Adaptive Server or ERSSD SQL Anywhere for this Replication Server
- ERSSD Replication Agent if a route is to be created from the current site
- Data servers with databases managed by this Replication Server
- Backup Server to back up Adaptive Server databases, including RSSDs
- Replication Servers at other sites that manage databases containing primary data that is replicated to this site
- Replication Servers at other sites with subscriptions for primary data maintained at this site
- Other Replication Servers to which this Replication Server has a route with no intermediate Replication Servers

You can use the default interfaces file or you can specify an alternative interfaces file at the command line when you start Replication Server. The interfaces file is usually located in the Sybase release directory. Use a text editor to modify the interfaces file. Refer to the Replication Server installation and configuration guides for your platform for more information.

Kerberos

Kerberos is a network authentication protocol that uses secret-key cryptography so that a client can prove its identity to a server in a network-based security mechanism.

Kerberos assumes the key distribution center (KDC) is running and configured for your realm, and the client libraries are installed under or on each client host in your realm.

Refer to the documentation and online help pages that are provided with Kerberos software for the configuration information.

Replication Server Principal Names

Replication Server accepts Kerberos Open Client™ connections that have a user-defined Replication Server principal name.

Replication Server authenticates the principal name with the Kerberos key distribution center (KDC). By default, the principal name is the name of the Replication Server. If you have multiple instances of Replication Server running, specify different principal names for each Replication Server.

To specify a different principal name:

- While starting Replication Server, set **-k rs_principal_name** option in the **repserver** executable program, or,

Replication Server Technical Overview

- Before starting the Replication Server, set the SYBASE_RS_PRINCIPAL environment variable in the **isql** command line tool.

Use **admin show_principal_name** command to view the Replication Server principal name.

If you modify the principal name of any Replication Server, execute **sysadmin principal_users[,reload]** to reload the principal name of all Replication Servers stored in the `rs_principal_users.cfg` configuration file.

See **admin show_principal_name** and **sysadmin principal_users[,reload]** in the *Replication Server Reference Manual*.

See also

- *Adaptive Server-to-Adaptive Server Replication with Kerberos Scenario* on page 266

Specifying the Replication Server Principal Name

To specify the Replication Server principal name, use either the SYBASE_RS_PRINCIPAL environment variable or the **-k rs_principal_name** option.

Prerequisites

To use Kerberos authentication with the principal names, configure these network-based security configuration files:

- `libtcl.cfg` on 32-bit machines
- `libtcl64.cfg` on 64-bit machines
- `objectid.dat`
- `interfaces` file

Task

1. Set the SYBASE_RS_PRINCIPAL variable before starting Replication Server:

```
setenv SYBASE_RS_PRINCIPAL <principal_name>
```

By default, the principal name is the name of the Replication Server. If you have multiple instances of Replication Server running, specify different principal names for each Replication Server. Replication Server uses the value of this variable to authenticate itself to Kerberos.

2. Set the **-k rs_principal_name**, before you start Replication Server using the **repserver** executable program:

```
-k rs_principal_name
```

When you start Replication Server with the Kerberos security mechanism enabled, Replication Server first uses the principal name specified with the **-k rs_principal_name** option for Kerberos authentication. If the **-k rs_principal_name** option is not specified, Replication Server looks for the principal name set in the SYBASE_RS_PRINCIPAL

environment variable. If neither is specified, Replication Server uses the server name for authentication.

In this example, the Replication Server name is "secure_rs" and the realm name is "MYREALM.COM". The Replication Server name is specified on the command line with `-s` parameter to the `dataserver`.

The current realm is specified in the `libtcl.cfg` configuration file on a 32-bit machine or `libtcl64.cfg` configuration file on a 64-bit machine by a `secbase` attribute value:

```
[SECURITY]
csfkrb5=libsybskrb.so
secbase=@MYREAL.COM libgss=/krb5/lib/libgss.so
```

To override the default Replication Server principal name:

- Option 1 – set the `-k rs_principal_name` option in the `repserver` program.

For example:

```
{repserver | repsvr} [-C config_file] [-i id_server]
[-S secure_rs] [-I interfaces_file]
[-E errorlog_file] [-M] [-v] [-K keytab_file] [-k rsprincipal]
[-upgr] [-A erssid_release_dir] [-purgeq]
[-nodb {all|dbid_1[,dbid_2[,dbid_3[,...]]}]
[-e]
```

where:

- `-s secure_rs@MYREALM.COM` – is the default Replication Server principal name.
- `-k rsprincipal@MYREALM.COM` – is the Replication Server principal name.

See *Replication Server Reference Manual* > *Executable Programs* > `repserver`.

- Option 2 – set `SYBASE_RS_PRINCIPAL` before starting Replication Server:

```
setenv SYBASE_RS_PRINCIPAL rsprincipal@MYREALM.COM
```

The Replication Server principal name that gets authenticated with Kerberos is "rsprincipal@MYREALM.COM", the value of `SYBASE_RS_PRINCIPAL` environment variable.

- Option 3 – if neither `-k rs_principal_name` nor `SYBASE_RS_PRINCIPAL` variable is set, Replication Server uses the `interfaces` file.

The Replication Server principal name that is authenticated with Kerberos is "secure_rs@MYREALM.COM".

See also

- *Configure libtcl.cfg* on page 263
- *Configure objectid.dat* on page 264
- *Configure the interfaces File* on page 265
- *Set Environment Variables (Kerberos)* on page 265

Considerations

Considerations for using the Kerberos principal name to access Replication Server.

- When you set the **use_security_services** parameter on and the **unified login** parameter to *required*, use these commands to access the Replication Server:

```
• isql -Vsecurity_option -Sserver_princ
```

```
• isql -Vsecurity_option -Sserver_princ -Uxx
```

In this case, you can access the Replication Server only with the credential of a principal user.

- When you set the **use_security_services** parameter on and the **unified login** parameter to *not_required*, use these commands to access the Replication Server:

```
• isql -Uxx -Pxx -Sserver_name
```

```
• isql -Vsecurity_option -Sserver_princ
```

```
• isql -Vsecurity_option -Sserver_princ -Uxx
```

In this case, you can access the Replication Server with the credential of a principal user or a password.

See also

- *isql Command Line Options for Security* on page 290

LDAP Server

An LDAP server provides global directory services for sharing component information such as server names and connection properties.

LDAP directory services allow components to look up directory information in a network-based system. Any type of LDAP service or gateway is an LDAP server. An LDAP driver calls LDAP client libraries to establish connections to an LDAP server. The LDAP driver and client libraries define the communication protocol and content of messages exchanged between clients and servers. LDAP runs directly over the Transmission Control Protocol (TCP).

When the LDAP driver connects to the LDAP server, the server establishes the connection based on one of two authentication models:

- Anonymous access – which does not require any authentication information, and is used typically for read-only privileges, or
- User name and password access – which is same as the user name and password used to access Replication Server.

Note: You can use an LDAP administrator account with a search filter for server lookup or authenticate a Replication Server user with an LDAP server.

Server Lookup

Replication Server uses the access information as an extension to the LDAP URL for server lookup.

Access information is taken from the `libtcl.cfg` file:

- UNIX – `$$SYBASE/$$SYBASE_OCS/config/libtcl.cfg`
- Windows – `%SYBASE%\%SYBASE_OCS%\ini\libtcl.cfg`

Replication Server uses LDAP for authenticating server names and connection properties. You can use LDAP services in place of an Replication Server interface file. The `libtcl.cfg` configuration file is used to search the server directory information. For example, Replication Server and Adaptive Server information.

You must edit the `ldap32` entry under the `DIRECTORY` section in `libtcl.cfg` to include `libsybdldap.so` and `libsybdldap64.so`. Otherwise, Replication Server cannot connect to LDAP on 64-bit platforms. For example:

```
[DIRECTORY]
ldap32=libsybdldap.so ldap://sylvester:3389
/dc=Sybase,dc=com??one??bindname=cn=Manager,dc=Sybase,
dc=com??secret
ldap64=libsybdldap64.so ldap://sylvester:3389
/dc=Sybase,dc=com??one??bindname=cn=Manager,dc=Sybase,
dc=com??secret
```

To start the LDAP session with the **dscp** utility, enter:

```
open ldap32
```

Replication Server uses Open Client/Server libraries to connect to LDAP servers and Open Client/Server configurations and procedures to set up and maintain LDAP services. See the *Replication Server Configuration Guide* for your platform for directions on how to set up an LDAP directory. For detailed information about Open Client/Server LDAP support, see the *Open Client Client-Library/C Reference Manual*.

LDAP User Authentication

Replication Server uses OpenLDAP client APIs to communicate with the LDAP server for user authentication.

The OpenLDAP API, which is precompiled with OpenSSL, is dynamically loaded from a shared library, `libsybaseldap*`, found in the `$$SYBASE/$$OCS/lib3p` or `$$SYBASE/$$OCS/lib3p64` directory.

The LDAP user authentication uses an LDAP URL to search the user information. The LDAP URL is defined in the **sysadmin ldap** command.

An example of LDAP URL:

```
ldap://john.doe.com:8888/dc=doe,dc=com??SUB?(cn=*)
```

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

Replication Server and LDAP User Account Management

An LDAP administrator creates and maintains user accounts in the LDAP server.

A database administrator creates and maintains Replication Server users. Alternatively, the database administrator can choose administration options that provide flexibility with user accounts when integrating Replication Server with the LDAP user authentication. For example, the database administrator administers the Replication Server roles, default database, language, and other login-specific attributes using administration commands and procedures.

Configure LDAP user authentication by setting the `user_authentication_source` parameter and the LDAP URL.

Table 2. Updates to rs_users if LDAP Authentication Is Enabled

Existing User in rs_users?	User Authenticated by LDAP?	Update to rs_users
No	Yes	Login fails. No change to rs_users.
No	No	Login fails. No change to rs_users.
Yes	Yes	Login successful. Update rs_users with the authenticated password.
Yes	No	<ul style="list-style-type: none"> • If user exists in LDAP – login fails. No change to rs_users. • If user does not exist in LDAP – login either fails or falls back to rs_users for decision with no changes.

LDAP User Authentication Configuration

The LDAP user authentication allows you to use LDAP enterprise-wide passwords instead of Replication Server passwords.

Replication Server uses the OpenLDAP API for the LDAP user authentication and this API supports LDAPv3.

When connecting to an LDAP server as a client, Replication Server supports any LDAP server, which conforms with a standard LDAP protocol, including Microsoft Windows Active Directory and OpenLDAP directory servers.

The primary data structure used with the LDAP protocol is the LDAP URL, which specifies a set of objects or values on an LDAP server. Replication Server uses LDAP URLs to specify an LDAP server, and search criteria to authenticate user login requests.

The LDAP URL uses this syntax:

```
ldapurl:=ldap://host:port/node?attribute?[base | one | sub]?filter
```

where:

- **host** – is the host name of the LDAP server.
- **port** – is the port number of the LDAP server.
- **node** – specifies the node in the object hierarchy at which to start the search.
- **attributes** – is a list of attributes to return in the result set. Each LDAP server may support a different list of attributes.
- **base | one | sub** – qualifies the search criteria.
 - **base** – specifies a search criteria for the base node.
 - **one** – specifies a search criteria for the base node and a sublevel under the base node.
 - **sub** – specifies a search of the base node and all node sublevels.
- **filter** – specifies the attribute or attributes to be authenticated. The filter can be simple, for example, "uid=" or compound, for example, "&(uid=*)(ou=group)".

The wildcard replacement fails if the first attribute in a wildcard search filter is not mapped to the login name attribute.

For example, if the search filter is "&(uid=*)(ou=group)" and the login name is "mylogin", during authentication, Replication Server uses "mylogin" to generate an output filter "&(uid=mylogin)(ou=group)". If the search filter is specified as "&(ou=*)(uid=*)", the login name is "ou=mylogin", which is incorrect. The wildcard replacement for the user authentication fails.

An example of the LDAP URL:

```
ldap://john.doe.com:8888/dc=doe,dc=com??SUB?(cn=*)
```

Upgrading Replication Server to Use LDAP User Authentication

Use **sysadmin ldap** and **configure replication server** to configure the LDAP user authentication.

Prerequisites

- Install, configure, and start the LDAP server using the vendor-supplied documentation.
- Add an **sa** user in the LDAP server.

Task

1. In upgraded Replication Server installations, configure LDAP user authentication.

- a) Test the LDAP server connection:

```
sysadmin ldap check_url, 'ldapurl' ['dn', 'pwd']
```

- b) Specify the LDAP server URL and values for an administrative access account:

```
sysadmin ldap  
set_primary_url, 'ldapurl'  
set_access_acct, 'dn', 'password'
```

where:

- **ldapurl** – is the primary LDAP server URL.
- **dn** – is the distinguished name (DN) of the administrative LDAP account. Each user entry in an LDAP server has a unique identifier called the distinguished name.
- **password** – is the password of an LDAP user.

- c) Configure the **user_authentication_source** to LDAP only:

```
configure replication server  
set user_authentication_source to 'ldap'
```

2. Add users for the replication system in the LDAP directory server using the vendor-supplied documentation.
3. Add the same set of users that you have added in step 2 using the **create user** command in Replication Server.

Note: If the LDAP user authentication is enabled, you cannot use the password specified with the **create user**, **set password** command. The password is synchronized from the LDAP server when the user is authenticated.

See *sysadmin ldap* and *configure replication server* in the *Replication Server Reference Manual*.

Migrating Existing Replication Server to LDAP User Authentication

Migrate the existing Sybase Replication Server to LDAP authentication using **sysadmin ldap** and **configure replication server** commands.

Prerequisites

Install, configure, and start the LDAP server using the vendor-supplied documentation.

Task

1. Test the LDAP server connection:

```
sysadmin ldap check_url, 'ldapurl'
```

2. Specify the LDAP server URL and values for an administrative access account:

```
sysadmin ldap
set_primary_url, 'ldapurl'
set_access_acct, 'dn', 'password'
```

where:

- **ldapurl** – is the primary LDAP server URL.
- **dn** – is the distinguished name (DN) of the administrative LDAP account.

3. Configure the `user_authentication_source` to both Replication Server and LDAP server:

```
configure replication server
set user_authentication_source to 'any'
```

4. Add all existing Replication Server users in the LDAP directory server.

You can add users simply by resetting the password for users in the LDAP server.

5. Set the `user_authentication_source` to LDAP only after adding all users to the LDAP server:

```
configure replication server
set user_authentication_source to 'ldap'
```

Note: If the LDAP user authentication is enabled, you cannot use the password specified with the **create user, set password** command. The password is synchronized from the LDAP server when the user is authenticated.

See *sysadmin ldap* and *configure replication server* in the *Replication Server Reference Manual*.

Configuring Primary LDAP Server to Use SSL/TLS

Configure the primary LDAP server URL to use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) for user authentication.

Replication Server supports two ways to secure the connection:

- Use SSL with an `ldaps://` URL.
- Use TLS with an `ldap://` URL and **sysadmin ldap, starttls_on_primary, starttls_on_secondary|starttls_on_secondary, 'true'** command.

1. Configure the Replication Server to use LDAP only for user authentication:

```
configure replication server set user_authentication_source to
'ldap'
go
```

2. Add a user to Replication Server and set the primary LDAP server URL with the certificate authorities (CA) root file path:

```
sysadmin ldap, set_primary_url,
'ldap://andre:888/dc=sybase,dc=com??sub?cn=*'
go
```

```
sysadmin ldap, set_access_acct, 'cn=Manager', 'secret'
go
```

Replication Server Technical Overview

```
sysadmin ldap, set_cacert_file,  
'user/SYBASE/config/trusted.txt'  
go
```

3. Enable TLS on the primary LDAP server and log in to Replication Server with a new user name and password:

```
sysadmin ldap, starttls_on_primary, 'true'  
go
```

```
isql -SmyRs -UnewUserName -PnewUserPwd
```

4. Configure the primary LDAP server to use the SSL port.

Before using this command, set the **starttls_on_primary** parameter to false.

```
sysadmin ldap, starttls_on_primary, 'false'  
go
```

```
sysadmin ldap, set_primary_url,  
'ldaps://andre:8889/dc=sybase,dc=com??sub?cn=*'  
go
```

5. Log in to Replication Server using the new user name and password:

```
isql -SmyRS -UnewUserName -PnewUserPwd2
```

Disabling LDAP User Authentication

Disable the LDAP user authentication in an existing Sybase replication system.

1. Configure the **user_authentication_source** to both Replication Server and the LDAP server:

```
configure replication server  
set user_authentication_source to 'any'
```

You must ensure that all passwords in `rs_users` are valid before you disable the LDAP user authentication. You can simply reset the password in Replication Server using the **reset password** command.

2. After completing the password validation for each user, configure the **user_authentication_source** to Replication Server:

```
configure replication server  
set user_authentication_source to 'rs'
```

See also

- *Resetting a Lost or Forgotten sa User Password* on page 250

Make Replication Server Connections

To connect data servers and Replication Servers at the sites on a LAN or WAN, the replication system administrator at each site defines connections and routes.

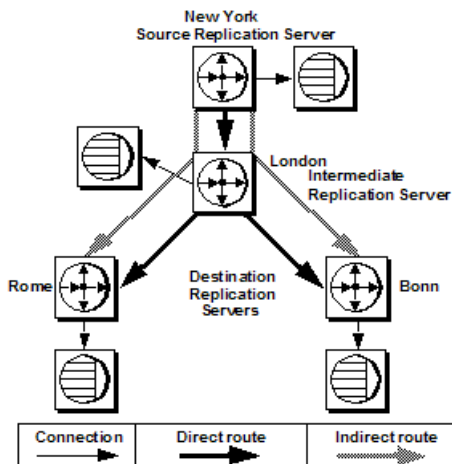
Organizing connections and routes is fundamental in planning replication. The connections and routes you establish determine the number of Replication Server components you need. In

addition, how you map replication between source and destination databases can impact system performance and data availability.

To specify where data is copied requires that you create the following paths or message streams between Replication Servers and between Replication Servers and databases in the system:

- A connection from a Replication Server to a database
Replication Servers distribute transactions received from primary databases through connections to the replicate databases they manage. A Replication Server may have connections to several databases, but each database can have only one connection from a Replication Server.
Warm standby applications also use a logical connection, which represents both a database and its standby database.
- A route from a Replication Server to another Replication Server
From each source Replication Server that manages databases containing primary data, you must specify a route to each destination Replication Server that subscribes to the data. You can specify a direct route from a source Replication Server to a destination Replication Server, or an indirect route, with intermediate Replication Servers between the source and destination Replication Servers.

Figure 8: Routes and Connections



This figure depicts an enterprise with several locations in Europe. A New York Replication Server routes all information for Europe through the London Replication Server. This arrangement reduces the number of direct connections the New York Replication Server makes and reduces WAN traffic. Data is sent once from New York to London, rather than from New York to each European location. The London Replication Server distributes the replicated data to the other European locations.

See the *Replication Server Design Guide* for details and rules on designing routes and connections for a replication system.

See *Replication Server Administration Guide Volume 2 > Managed Warm Standby Applications* for more information about logical connections.

See also

- *Manage Routes* on page 165
- *Manage Database Connections* on page 189

Specify Database Operations

Replication Server distributes database operations from a primary database to destination Replication Servers as functions that consist of a name and a set of data parameters.

The destination Replication Server then uses function strings to map functions to the commands recognized by the destination data server. These commands represent transaction-control directives (**begin transaction** or **commit transaction**) or data-manipulation instructions (**insert**, **update**, or **delete**). The function string serves as a template or meta-command that transforms a function to a data-server-specific command. The use of function strings makes it possible for a primary site to replicate data to multiple heterogeneous data servers. Function strings are categorized into function-string classes according to data server type.

For example, a primary Replication Server transmits the **rs_insert** function to a destination Replication Server, which uses the appropriate function string to translate the function into the insert command specific for the data server in use at that site, whether the database is Adaptive Server, DB2, or another database.

There are two types of functions:

- System functions – represent data-server operations with function strings supplied by Replication Server or available when you install a new database to the replication system.
- User-defined functions – allow you to customize Replication Server applications to distribute stored procedures.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations* for details.

Function Strings

Function strings for functions can be automatically generated for function-string classes that come with Replication Server.

Function strings must be customized for any function-string class that the user creates that does not inherit its functions strings from one of the provided classes. To customize a function string, you modify an existing function string with data-server-specific commands or by

invoking a remote procedure call (RPC). A customized function string can also contain function string variables that represent the values of columns, procedure parameters, system-defined information, and user-defined variables. Replication Server replaces the variables with actual values before sending function strings to the data server.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations* for details.

Function-string Classes

A function-string class comprises all of the function strings used with a type of database.

Replication Server provides function-string classes for Adaptive Server and with version 15.2, Replication Server provides function-string classes for non-ASE data servers: Sybase® IQ, HANA DB, IBM DB2 UDB, Microsoft SQL Server, and Oracle. Although function strings may contain data-server-specific instructions, they can often be used with several databases maintained by the same data server type. You can create classes with all new function strings or create a derived class that inherits function strings from an existing parent class.

Transaction Handling with Replication Server

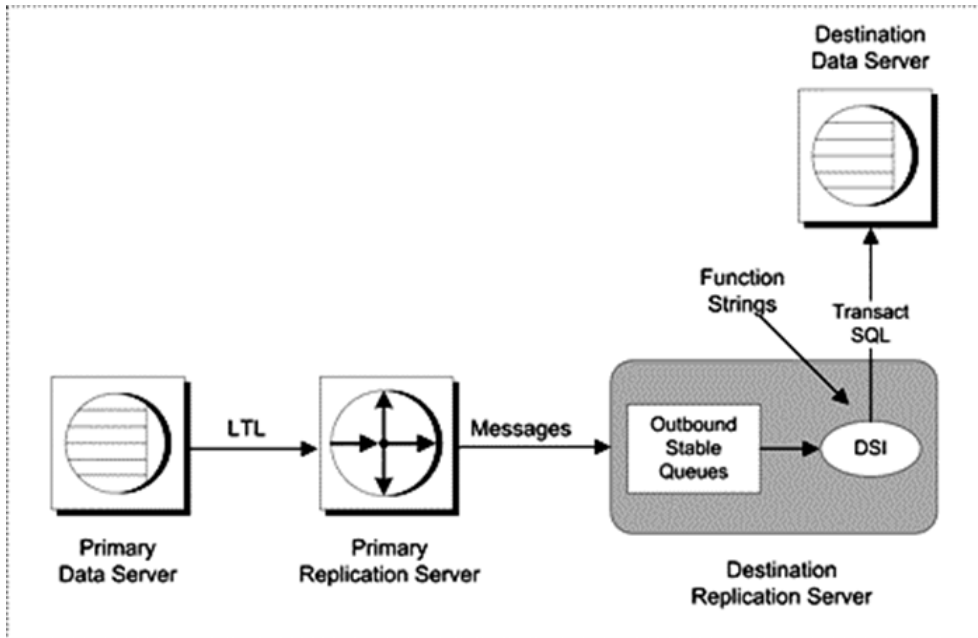
Replication Server depends on data servers to provide the transaction-processing services needed to protect stored data.

To ensure the integrity of distributed data, data servers must comply with the following transaction-processing conventions:

- A transaction is one unit of work. Either all operations in the transaction are performed or none are performed.
- Transaction results are permanent. A transaction cannot be arbitrarily undone after it is committed.

Replication Server copies committed transactions from primary sites to replicate sites, and distributes transactions in the order they are committed so that copied data passes through the same states as the primary data.

This figure illustrates the Replication Server method for translating transactions.



Once the primary Replication Server sends transactions to subscribing sites, destination Replication Servers store the transactions in the outbound Data Server Interface (DSI) stable queue.

Stable Queues

When you install Replication Server, you set up a disk partition that Replication Server uses to establish stable queues. During replication operations, Replication Server temporarily stores updates in these queues.

There are three types of stable queues, each of which stores a different type of data:

- Inbound queue – holds messages only from a Replication Agent. If the database you add contains primary data, or if request stored procedures are to be executed in the database for asynchronous delivery, Replication Server creates an inbound queue and prepares to accept messages from a Replication Agent for the database.
- Outbound queue – holds messages for a replicate database or a replicate Replication Server. There is one outbound queue for each of these destinations:
 - For each replicate database managed by a Replication Server, there is a Data Server Interface (DSI) outbound queue.
 - For every Replication Server to which a Replication Server has a route, there is a Replication Server Interface (RSI) outbound queue.
- Subscription materialization queue – holds messages related to newly created or dropped subscriptions. This queue stores a valid transactional “snapshot” from the primary

database during subscription materialization or from a replicate database during dematerialization.

You can add more partitions later if your replication system requires more space for stable queues. You can also create partitions that automatically grow or shrink according to usage.

See the *Replication Server Troubleshooting Guide* for information on how to examine queue contents for troubleshooting purposes.

See also

- *Partitions for Stable Queues* on page 53
- *Automatically Resizable Partitions* on page 54

Queue Management

Replication Server uses the Stable Queue Manager (SQM) thread to manage each stable queue.

Threads are subprocesses that manage specific tasks, such as receiving messages. Some queues also have an additional Stable Queue Transaction (SQT) thread. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Replication Server Internal Processing > Processes in the Primary Replication Server* for details on the SQM and SQT threads.

When transactions are ready to leave the stable queue, one of these threads submits the transactions in the queue:

- Data Server Interface (DSI) thread – manages the connection with the data server.
- Replication Server Interface (RSI) thread – manages the connection with the replicate Replication Server.

The enhanced queue **dump** commands give you flexibility in identifying the stable queues, controlling the stable queue contents to dump, and supporting additional output file options. In addition, Replication Server introduces commands that allow you to delete and restore specific transactions from the SQM.

Commands to Manage Stable Queues

Replication Server provides RCL commands to manage stable queues.

- **sysadmin dump_queue**
- **sysadmin sqt_dump_queue**
- **resume connection**
- **sysadmin log_first_tran**
- **sysadmin sqm_zap_tran**
- **sysadmin sqm_unzap_tran**
- **sysadmin dump_tran**

See the *Replication Server Reference Manual* for detailed information about these commands.

DSI Thread

The DSI thread translates the transaction modifications into RPCs or the language as specified by the function strings in the function-string class assigned to the destination database.

Replication Server starts DSI threads to submit transactions to a replicate database to which it has a connection.

The DSI thread performs the following tasks:

- Collects small transactions into groups by commit order.
- Maps functions to function strings according to the function-string class assigned to the database connection.
- Executes the transactions in the replicate database.
- Takes action on any errors returned by the data server; depending on the assigned error actions, also records any failed transactions in the exceptions log.

To improve performance in sending transactions from a Replication Server to a replicate database, you can configure a database connection so that transactions are applied using multiple DSI threads. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI threads* for a description of this feature.

Replication Server version 15.2 introduces support for bulk copy-in to improve performance when replicating large batches of **insert** statements on the same table in Adaptive Server® Enterprise 12.0 and later. Replication Server implements bulk copy-in using the Open Client™ Open Server™ Bulk-Library. See *Replication Server Administration Guide Volume 2 > Performance Tuning > DSI Bulk Copy-in*.

The DSI thread may apply a mixture of transactions from all data sources supported by the Replication Server. The transactions are processed in the single outbound stable queue for the destination data server.

RSI Thread

RSI threads send messages from one Replication Server to another. There is one RSI thread for each destination Replication Server.

The primary Replication Server processes transactions, causing those destined for other Replication Servers to be written to RSI outbound queues. An RSI thread logs in to each destination Replication Server and transfers messages from the stable queue to the destination Replication Server.

When a direct route is created from one Replication Server to another, an RSI thread in the source Replication Server logs in to the destination Replication Server. When an indirect route is created, Replication Server does not create a new stable queue and RSI thread. Messages for indirect routes are handled by the RSI thread for the direct route.

See also

- *Establish Replication Server Connections* on page 36

Partitions for Stable Queues

Replication Server stores messages destined for data servers or other sites on partitions.

Replication Server allocates space in a stable device by partition and further divides the partitions into segments and blocks. Each stable queue holds messages to be delivered to another Replication Server or to a database.

The **rs_init** program assigns the initial partition to the Replication Server. Refer to the Replication Server installation and configuration guides for more information about working with partitions in **rs_init**.

The minimum initial partition is 20MB. You may need additional partitions, depending on the number of databases the Replication Server manages and the number of remote sites to which the Replication Server distributes messages. Larger partitions may also be necessary when subscriptions are initiated or when there are long-running transactions.

A Replication Server can have any number of partitions of varying sizes. The sum of the partition sizes is the Replication Server capacity for queued transactions.

Use the **create partition** command to assign additional partitions or **alter partition** to expand partitions. See the *Replication Server Reference Manual* for syntax, parameter descriptions, examples and usage information.

You can also create automatically resizable partitions that grow or shrink according to usage.

When choosing a partition for Replication Server, consider these guidelines:

- For the best performance, install Replication Server partitions on fast file systems or operating system raw devices.

Restriction: You cannot use raw devices on Windows or for automatically resizable partitions.

- Do not mount the partition for use by the operating system.
- Do not use the partition for any other purpose, such as storing file systems, maintaining swap space, or locating Adaptive Server devices.
- Allocate the entire partition to Replication Server. If you allocate just a portion of a partition for Replication Server, you cannot use the remainder for any other purpose.
- Do not allow any users read/write permissions on the partition unless the user is going to start Replication Server.
- Since the **sqm_async_seg_delete** parameter that you can set with **configure replication server** is on by default, Replication Server may require a larger partition when you upgrade to version 15.7 or later. See:
 - *Replication Server Configuration Guide > Preparation for Installing and Configuring Replication Server > Plan the Replication System > Initial Disk Partition for Each Replication Server.*

- **sqm_async_seg_delete** and **alter partition** in *Replication Server Reference Manual > Replication Server Commands*.

You can choose how Replication Server allocates queue segments to partitions or you can use the default mechanism. The default mechanism assigns queue segments to the next partition in an ordered list. Use the **alter connection** or **alter route** command to choose a different allocation mechanism. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Allocate Queue Segments* for more information.

See also

- *Automatically Resizable Partitions* on page 54

Use Disk Files for Stable Queues

To use a disk file for a partition, create the file before you execute the **create partition** command.

You can create an empty file and set its permissions so that Replication Server can read and write to the file. Replication Server extends the file to the size you specify.

Partitions can be either raw disk partitions, which is preferable, or operating system files. Where a choice is available, raw disk partitions provide the best recoverability, since disk writes to raw disk partitions are not buffered by the operating system.

Automatically Resizable Partitions

Use Replication Server partitions that automatically grow or shrink according to usage.

An automatically resizable partition consists of separate partition files. With an automatically resizable partition, you need not manually increase the partition size or create a new partition when consumption from storing stable queue messages increases, and you need not manually reduce the partition size to conserve disk space if consumption decreases.

Instead, Replication Server responds immediately and automatically grows by creating a new partition file when consumption reaches 80% of the total capacity of the partition. You can set the individual partition file size according to your requirements, and you can set a limit to the total size of the partition according to the available disk space. Replication Server does not create a new partition file, even if consumption for the entire partition reaches 80% capacity, if Replication Server calculates that a new file exceeds the limit for the total size of the partition.

In addition, Replication Server automatically conserves disk space if consumption of the partition drops, by removing the last partition file that was added in a automatically resizable partition, if the file becomes empty. To improve response time and replication performance in case there are rapid fluctuations in disk consumption, Replication Server delays the shrinking of the partition until the total partition usage without the last partition file is lower than 50%.

Use **create auto partition path** to:

- Create a logical partition path for the partition files that Replication Server automatically creates.
- Associate the logical partition path to a physical location with sufficient disk space for the partition files.
- Set the size of the partition files that Replication Server automatically creates.
- Set the maximum total size for all the partition files in the partition.

```
create auto partition path logical_name
on 'physical_path'
with auto expand size = size
max size = max_size
```

For example, to create the `auto_uxp` logical partition path on the UNIX device named `/usr/user1`, with an initial partition file size of 100MB, and automatically add a new 100MB partition file to `auto_uxp` every time the partition file reaches 80% usage, and limit the total size for all the partition files that Replication Server can create automatically to 102,400MB, enter:

```
create auto partition path auto_uxp on '/usr/user1'
with auto expand size=100 max size=102400
```

You can create multiple automatically resizable partitions but each partition must have a unique physical path or logical name, and sufficient disk space.

Replication Server names the partition files that it automatically creates according to the "`logical_name`"_"`partition_number`" format, where *partition_number* is a ten-digit number that Replication Server generates automatically and increases sequentially from 0000000001 to 2147483647 with each file that Replication Server adds. Replication Server can automatically grow or shrink partitions only with partition files named using this format. All other partitions must be manually managed by Replication Server administrators.

Warning! Do not manually delete files in the partition physical location with names that follow the *logical_name_partition_number* format as the partition files may contain data, unless the partition file has not been used by Replication Server. You can use **admin disk space** to check if Replication Server is using the partition file. Consult the system administrator or technical support.

To manage automatically resizable partitions, use:

- **alter auto partition path** – to change the size of one partition file and the maximum size of a automatically resizable partition
- **drop auto partition path** – to remove an automatically resizable partition from Replication Server
- **admin auto_part_path** – to display information on automatically resizable partitions
- **admin disk space** – to check if a device is an automatically resizable partition
- **rs_helppartition** – stored procedure to display information on all automatically resizable and manually managed partitions

See the *Replication Server Reference Manual* for descriptions and examples of all the commands and the stored procedure.

There are several restrictions and guidelines to consider when you create and manage automatically resizable partitions

- You must have write permission to the physical path before you can create a automatically resizable partition.
- You cannot create a automatically resizable partition on a raw device.
- You cannot use the **disk_affinity** parameter to select a automatically resizable partitions for partition affinity. See *Select Disk Partitions for Stable Queues* in the *Replication Server Administration Guide Volume 2*.
- You can use **drop auto partition path** to remove only automatically resizable partitions. You cannot use **drop auto partition path** to remove partitions created with other commands.
- You can use **drop partition** to manually remove an automatically resizable partition if there is an urgent need to release disk space. Otherwise, use **drop auto partition path** to manage the removal of automatically resizable partitions to ensure that disk space is released properly.
- Replication Server does not immediately remove an automatically resizable partition if you execute **drop auto partition path**. Instead, once this command marks the affected logical partition path as "drop-pending", Replication Server does not create any new automatically resizable partition file on the path. Replication Server only removes the automatically resizable partition after the partition files on the path have been dropped by Replication Server.
- Installation of Replication Server creates an initial partition. The initial partition and any partitions you subsequently create manually constitute the minimal partition size that Replication Server retains even after you remove all automatically resizable partitions. You can remove some of the partitions in the minimal partition size but you must ensure that Replication Server retains some partitions of a sufficient size for replication to continue.

Distributed Concurrency Control

Data servers that store primary data provide most of the concurrency control needed for the distributed database system. If a transaction fails to update a primary version of a table, the primary Replication Server does not distribute the update to other sites.

When a transaction succeeds in updating primary data, the Replication Server distributes the changes. Unless a failure occurs, the update succeeds at all sites with subscriptions to the data.

Transactions that Modify Data in Multiple Databases

A transaction that modifies primary data in more than one data server may require additional concurrency control.

According to the transaction processing requirements, either all of the operations in the transaction must be performed, or none of them. If a transaction fails on one data server, it must be rolled back on all other data servers updated in the transaction.

If a multi-database transaction is replicated, updates to each database flow to replicate databases as independent transactions because there is one Replication Agent per database.

Failed Replicate Table Updates

A modification to primary data may fail to update a copy of the data at a subscribing site.

The primary version is the “official” copy and updates that succeed there are expected to succeed at subscribing sites with copies. If the updates do not succeed, one of the following reasons may explain why:

- Replicate and primary versions are out of sync following a system recovery and a loss has been detected.
See [Replication Server Administration Guide Volume 2 > Replication System Recovery](#) for more information.
- The data server storing the copy of the table has constraints that are not enforced by the data server storing the primary version.
- The data server storing the copy of the table rejects the transaction due to a system failure, such as lack of space in the database or a full transaction log.

When a transaction fails, Replication Server records the transaction in an exceptions log for handling that is appropriate to the application. Replication Server offers error handling flexibility through its error action feature. This feature allows responses to data server errors based on your own defined configuration settings. For example, you can specify that transactions be retried at the site where they failed.

A client at each site must resolve transactions in the exceptions log, because the appropriate resolution is application-dependent. In some cases, you can automate the resolution by encapsulating the logic for handling rejected transactions in an intelligent application program.

Transaction Processing by the Replication Agent

The Replication Agent scans the database transaction log and sends transaction information to Replication Server for distribution to subscribing databases. Learn the transaction processing by Adaptive Server RepAgent thread.

Note: Replication Agents for other databases may work differently. See the Replication Server Options documentation for the Replication Agent and database you are working with.

Coordinate Adaptive Server Log Truncation

As long as there is space in the Adaptive Server database transaction log, Adaptive Server continues to process transactions. To prevent the log from filling up, it must be emptied (“truncated”) periodically.

You can use the Adaptive Server **dump transaction** command or set the Adaptive Server **trunc log on chkpt** option to “on” so that the log truncates automatically.

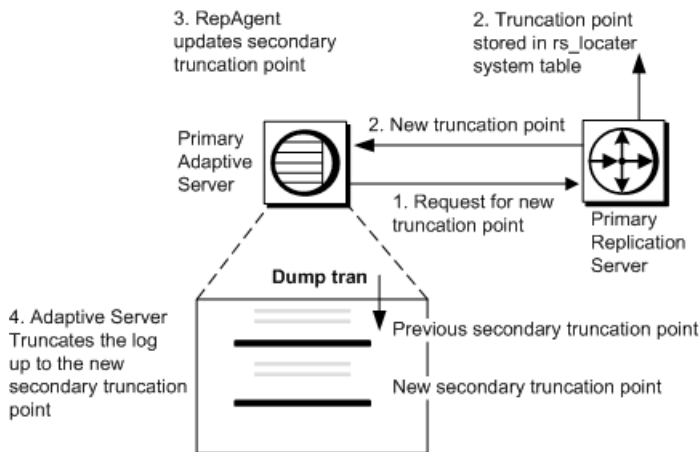
Each primary database maintains primary and *secondary truncation points* in its database log. The primary truncation point marks the last log record Adaptive Server has finished processing. The secondary truncation point normally marks the log record that contains the **begin transaction** command for the oldest open transaction not yet fully applied by Replication Server. Replication Server stores a copy of the latest secondary truncation point in the `rs_locator` table of the RSSD.

RepAgent requests a new secondary truncation point when it has scanned a predetermined number (batch) of records or has reached the end of the log and there is no new activity. Replication Server acknowledges receipt of a batch of transaction records by giving RepAgent the information that allows it to move the secondary transaction point.

Adaptive Server makes sure that only transactions already processed and passed to the Replication Server are deleted by never truncating the log past the secondary truncation point.

RepAgent updates the secondary truncation point as shown in the "Adaptive Server Log Truncation" figure.

Figure 9: Adaptive Server Log Truncation



1. RepAgent requests a new secondary truncation point from the primary Replication Server.
2. The primary Replication Server returns the latest secondary truncation point to the RepAgent and also writes it into the `rs_locator` system table.

3. RepAgent updates the secondary truncation point in the transaction log.
4. At the next checkpoint or **dump transaction** command, the log is truncated up to the new secondary truncation point.

Schema information describes the structure of the database. Each time you change the schema of a database object—such as dropping a table, creating a clustered index, or renaming a column—Adaptive Server records current schema information for that object. Thus, when RepAgent scans the transaction log, it can always retrieve the correct schema for a table or procedure—even if the original database object has been changed or no longer exists. You do not need to drain the transaction log before executing schema changes at the primary site.

Manage Replication Environment with Sybase Central

You can manage your replication environment using the Replication Manager (RM) plug-in and Replication Monitoring Services (RMS).

Sybase integrates its systems management tools into one desktop product, called Sybase Central. Each server product, such as Replication Server or Adaptive Server, can be managed from Sybase Central.

After installing and configuring Replication Server, start Sybase Central, and use RM to create a new replication environment that includes all data servers, Replication Servers, and Replication Agents that participate in the replication.

Replication Manager displays a two-pane window that contains icons for the servers managed in the environment. Use this window to monitor the status of the servers and to execute menu commands to diagnose and manage the servers and other components of your replication system.

See also

- *Replication System Components* on page 23

Start and Stop Sybase Central

Learn how to start and stop Sybase Central.

Starting Sybase Central

You can start Sybase Central using multiple methods.

To start Sybase Central:

- On Windows, do any of the following:
 - Select **Start > Programs > Sybase > Sybase Central v6.0.**
 - Create a shortcut on your desktop for Sybase Central.
 - Double-click `scjview.exe` in `%SYBASE%\Shared\Sybase Central 6.0.0/win32`.
 - Add Sybase Central to your Startup program group.
- On UNIX, execute `scjview.sh` in `$SYBASE/shared/sybcentral600`.

Stopping Sybase Central

To stop Sybase Central, select **File > Exit**.

Online Help

The Replication Manager plug-in provides online help for performing specific tasks through Sybase Central.

The different types of help in the Replication Manager plug-in include:

- Topic help
- Tooltips
- Status bar messages

Topic Help

Topic help describes how to use Sybase Central to manage your replication system. To display topic-level help, select Help from the Sybase Central main menu, then select Replication Manager Online Help.

The Replication Manager help browser opens with two panes. The left pane displays the table of contents, and the right pane displays contents of the selected topic.

Click the Contents tab to browse through the topics by category.

- Book icons represent headings. Double-click a book icon to see the sub entries under that heading. Sub entries can be other book icons or page icons. Topic headings are organized around Replication Server concepts (for example, Managing Users, Managing Database Connections, and others) for easy reference.
- Page icons represent topics that describe tasks, or concepts that correspond to the heading under which they are listed. Topics are generally organized in the order that you would perform procedures under that heading. Double-click a page icon to display a topic.

Tooltips

Tooltips are small pop-up windows that provide a description of a control (that is, a toolbar button or menu option) when a pointer is moved over that control.

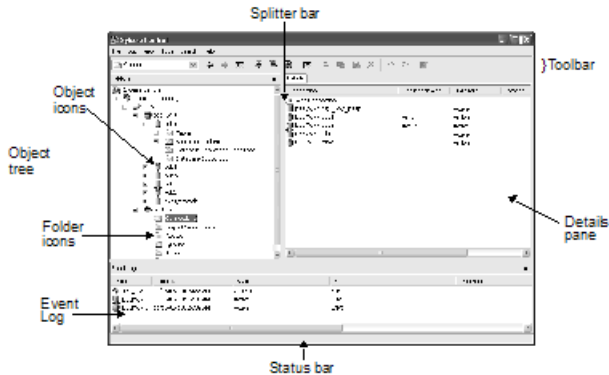
Status Bar

The status bar is an information display bar located at the bottom of the application window. In Sybase Central, the status bar displays a brief description of the menu command at which the cursor is currently pointed. The help line appears on the left side of the status bar.

Replication Manager User Interface

The Replication Manager user interface that displays within the Sybase Central framework allows you to access replication environment and server objects.

Figure 10: Replication Manager in Sybase Central Main Window



The main window is divided into left and right panes. When you are connected to a replication environment:

- The left pane displays a hierarchical list, or object tree, which shows:
 - Icons for folders and objects in the replication environment
 - Icons for other plug-ins, such as the Sybase Central plug-in for Adaptive Server Enterprise, if they are installed
- The right pane displays the contents of the folder or object selected in the left pane.

To adjust the size of the panes, use the mouse pointer to drag the splitter bar to the left or right.

Use Replication Manager

Folder and Object Icons

The main window includes folder icons and object icons.

Each folder icon contains all objects of its type within the replication environment. For example, the Connections folder shows all connections. Folder icons can appear in either the left or right pane.

Note: A Replication Server connection is associated with one database in an Adaptive Server.

Manage Replication Environment with Sybase Central

Each object icon represents one server object, such as a database, a table, a replication definition, or a connection. Some objects contain other objects, such as a database containing tables.

To select an object, click its icon. The type of object selected determines the range of commands available. For most activities, you must select an object before you can perform any operation on it.

Details Tab

When you select an object in the left pane, one or more tabs appear in the right pane. For most objects, a single tab called “Details” is displayed, containing a list of general information about the object.

The Details list displays:

- Sub components, which are replication or database objects that are contained in another object.
- Function components, which invoke a wizard when double-clicked.

Note: When you select a thread object, several tabs appear in the right pane.

Move Through the Sybase Central Object Tree

You can use different techniques to see different parts of the object tree.

To move vertically through the current display, use the scroll bar on the left or right pane.

To expand or collapse the list to show different levels of detail, do one of the following:

- Click the plus or minus buttons. A plus button next to an icon indicates that the list of objects for that icon can be expanded. A minus button indicates that the list of objects for the icon is fully expanded.
- Double-click a folder icon, which expands the list in the right pane and changes the view to a list of objects in the folder. For most objects, double-clicking an object icon in the right pane opens a property sheet that displays information.

Customize the Display

You can customize the display to hide or redisplay the folders, toolbar, status bar, or event log, select Folders, Toolbar Status Bar, or Event Log from the View menu.

To move the display of the right pane tabs from top to bottom or from left to right, select **Tools > Options**.

Keyboard Shortcuts

In addition to using a mouse, you can use keyboard shortcuts to choose menu commands and navigate through dialog boxes.

Every menu title and menu command has an underlined letter, called a *mnemonic*. To select a menu, press **Alt+mnemonic**. To choose a menu command, press the mnemonic key. You can

execute some commands directly by pressing Ctrl plus another key, or by pressing a function key. These shortcuts are listed on the menus.

To navigate to different controls (for example, fields, lists, and buttons) in a dialog box or property sheet, use the Tab key. To select different tabs in a property sheet, use the Tab key to select the current tab, then use the left and right arrow keys to select other tabs.

Context Menu

A context menu is specific to the selected object and contains commands that are executed against the selected objects.

You can execute some commands against multiple objects at once. To activate a context menu, right-click an object icon. From the menu that appears, choose the appropriate command.

Toolbar

The Sybase Central toolbar provides quick alternatives for executing frequently used menu commands.

The toolbar has the following controls:

- A drop-down list box that displays the hierarchy of the currently selected object. You can select an object higher up the hierarchy to change the focus of the main window.
- Buttons, which provide a quick way to execute menu commands.

Toggle Toolbar on the View menu to display or hide the toolbar.

Status Bar

The status bar is an information display bar located at the bottom of the application window. In Sybase Central, the status bar displays a brief description of the menu command at which the cursor is currently pointed. The help line appears on the left side of the status bar.

Toggle Status Bar on the View menu to display or hide the status bar.

Event Log Pane

The Replication Manager displays the Event Log pane, which shows events that occur in the replication environment.

These events can be:

- Component state changes for connections, routes, and queues
- Server availability changes
- Background thread completion
- Background processes status
- RMS event trigger execution

Background Processes

You can monitor background replication processes and stop these processes.

Several tasks performed by the Replication Manager can be very time-consuming, such as creating a subscription that also materializes the table. These tasks are performed in the

background, allowing Replication Manager to perform other tasks. When you start a time-consuming task, Replication Manager displays a message window to indicate a running process. Click **Stop Process** to cancel the background process or click **Close** to close the window, and continue running the process in the background.

When a background task is completed, Replication Manager puts an event entry in the event log.

To see the status of a background process at a later time, open the Background Processes dialog, which displays a list of all the currently running processes.

Select **Search > Background Processes** to access the Background Processes window which displays:

- Process – the name of the process.
- Start Time – the start time of the process.
- Status – the status of the process.

Script Editors

You can use the script editors to view generated RCL commands, which include syntax to create any objects such as connection, routes, and replication definitions.

Replication Manager provides two script editors; the Replication Command Language (RCL) script editor and the Structured Query Language (SQL) script editor. Both editors operate in the same way, except the RCL script editor highlights the RCL keywords while the SQL script editor highlights the SQL keywords.

Accessing the Script Editor

Learn how to access the script editor.

1. Select the Replication Server object for which you want to generate RCL.
2. Right-click that object.
3. Select Generate RCL from the context menu. The selected script editor window opens and contains the RCL needed to create the selected object.

Monitor Status

Replication Manager graphically displays the status of the servers and components in the environment.

Replication Manager shows an object icon, which changes depending on the state of an object. The status of the servers, connections, routes, and queues also displays in the Properties dialog.

The status of an environment is the state of its components. The status of a server or component includes its current state and a list of reasons that describe the state.

As you work, information in open dialog boxes and the Sybase Central window may become unsynchronized. To update the contents of the window, select **View > Refresh Folder** or **View > Refresh All**, or press **F5**.

Hiding Connection Status

You can hide (or filter out) the connection status if you do not want to see the status either on the individual connection icon or as part of the rollup status for Replication Server.

The options for hiding connection status are as follows:

- Hide the State of the Replication Agent — hides the state of the Replication Agent thread in the Details list, on the Connection Properties dialog box, and in the rollup status for the Replication Server to which that Replication Agent thread is connected.
- Hide the State of the DSI Thread — hides the state of the DSI thread in the Details list, on the Connection Properties dialog box, and in the rollup status for the Replication Server to which the DSI thread is associated.

To hide the connection status:

1. Right-click the connection for which to hide the status.
2. Select Hide Connection Status from the drop-down menu.

A dialog box displays which shows options for hiding the connection status.

3. Select an option.

The state for the connection now reads “Hidden.” The state on the Connection Properties dialog box and in the rollup status for the Replication Server is also hidden. The Event Log records this change.

Filter Connection Status in Different Instances of Replication Manager

The filtering state of a connection status is stored locally by the Replication Manager.

Therefore, different instances of Replication Manager do not share filtering states. For example, if you create a connection using one instance of Replication Manager, and then set the Replication Status to hide for that connection, another Sybase Central plug-in instance monitoring the same environment does not filter the connection status; filtering information is available only to the original Replication Manager instance.

In addition, any connection created outside of Sybase Central (by **rs_init** or from the command line) is not filtered automatically by the Replication Manager. You must set the filtering manually from within Sybase Central.

Filter Connection Status in Warm Standby Environments

If you are creating a warm standby environment, the Replication Manager automatically sets the filtering state for the active Data Server Interface (DSI) thread and standby RepAgent thread connections.

You must set filtering for the physical connection manually by selecting one of the connection status hide options from the context menu.

Perform Common Tasks

Creating an Object

You can create replication definitions, subscriptions, connections, and other Replication Server objects in Sybase Central.

1. Select the folder for the type of object you want to create.
2. Select **File > New**.
3. Select the object name. One of the following occurs:
 - If a wizard exists to help you create the object, the wizard opens. Respond to the wizard prompts.
 - If no wizard exists, a property sheet displays. Fill in the information for the new object.

Object Properties

After you create an object, it is represented by an icon in any pane of the Sybase Central window. You can display or update the object by opening its Properties dialog.

A Properties dialog contains information about the object and how it relates to other objects in the replication environment. The Properties dialog can also provide a direct navigation path to its related objects. It enables you to enter information for a new object on tabbed pages.

The Properties dialog box generally has three tabs.

- General tab – displays all status.
- Communications tab – displays information on how Replication Manager communicates with the server.
- Parameters tab – displays and allows modification to configuration parameters for servers and components.

Note: Some Properties dialogs may have different tabs. For example, a connection has General, Security, and Parameters tabs.

Viewing Object Properties

To view object properties in Sybase Central, select the object icon and then select **File > Properties**.

Deleting an Object

You can delete replication definitions, subscriptions, connections, and other Replication Server objects in Sybase Central.

1. Select the object icon.
2. Select **Edit > Delete**.
3. Confirm the deletion in the confirmation dialog.

Data Server Names in Replication Manager

Data servers in Replication Manager must have unique names, and the names of the non-Sybase data servers must match the Replication Agent configuration parameter **rs_source_ds**.

If an existing environment uses the same name for the Replication Agent and the configuration parameter, change the name of the agent by manually adding the server name, host, and port number in page 3 of the Add Server wizard.

Set Up a Replication Environment

A replication environment includes replication objects such as Replication Servers, data servers, and Replication Agents. Before any replication activities can be performed, you must create and configure an environment.

Depending upon the scale and complexity of your replication environment, you can set up either a two-tier or a three-tier management solution for your environment.

Two-tier Management Solution

In a two-tier management solution, Replication Manager (RM) connects directly to the servers in the environment without communicating through a management layer.

This lets you manage small, simple replication environments with fewer than 10 servers, and provides you the ability to create, alter, and delete components in the replication environment.

Three-tier Management Solution

In a three-tier management solution, Replication Manager can monitor larger and complex replication environments with the help of Replication Monitoring Services (RMS). RM connects to the servers in the environment through RMS.

RMS provides the monitoring capabilities for the replication environment. In this solution, RMS monitors the status of the servers and other components in the replication environment, and RM provides the client interface that displays the information provided by the RMS.

Preparing for a Two-tier Solution

You must complete several steps to prepare your replication environment for a two-tier management solution.

Note: Replication Manager does not require a Sybase interfaces file, but you have the option to use it.

1. Install the Replication Server and Sybase Central software. See the release bulletin and installation guide for your platform.
2. Identify the data servers to be used in your replication system. If the data servers are not yet installed, do so using the installation guide for your specific data server.
3. Use **rs_init** to configure the Replication Server. See the *Replication Server Configuration Guide* for your platform.
4. Start Sybase Central.
5. In Sybase Central, create a replication environment and add the data servers and Replication Servers.

Create an Environment

Creating the replication environment involves giving a name to your environment, creating an environment object, setting up permissions, and adding servers.

Creating a Replication Environment Object

Use Replication Manager to create a replication environment object.

1. In Sybase Central, click the **Replication Manager** icon in the left pane of the Replication Manager window. The **Add Replication Environment** icon displays on the Details pane.
2. Double-click the Add Replication Environment icon in the right pane.
3. Enter the name of your environment and click **Next**.
4. Enter a user name and password that will let you access the environment. Click **Next**.

Note: Java does not support the roman8 character set. Do not select roman8 in the character set field for a server connection. Do not select “default” in the character set field for a server connection if the server default is roman8. You must select another character set that is compatible with the server and Replication Manager.

5. From the list of servers, select the ones to add to your environment, then add a user name and password for each. If you are adding a Replication Server, enter a user name and password for the RSSD. Click **Next**.

When adding servers, you must provide a user name and password that have been granted certain permissions:

- Replication Server – **sa** permissions.

- Adaptive Server Enterprise – the **sa_role** and the **sso_role**.
- Replication Server RSSD – the database owner.

Note: You can either select a server from the list or enter a server name, host, and port number. This list is from the interfaces file found in the \$SYBASE directory.

6. Check the summary page to make sure you have added all the servers you need. Then click **Finish**.

Note: You are not required to add all servers when you create the environment. You can add new servers to an existing environment by using the Add Server wizard.

The new environment object displays in the left pane under the Replication Manager object with the name you assigned.

Note: If you update the interfaces file while Sybase Central is running, you need only restart the wizard, or reopen any dialog box in progress. You need not restart Sybase Central for the changes to take effect.

Dropping a Server from a Replication Environment in Sybase Central

Use Replication Manager to drop a server from the replication environment.

1. Select the server you want to drop.
2. Do one of:
 - Click the **Delete** icon from the toolbar.
 - Right-click the selected server, select **Delete**.

Note: Although Sybase Central removes the server from the replication environment's server list and removes the server icon from the environment, the server is not removed from your replication system. If there are routes or database connections still associated with a deleted server, the server name may still appear in dialog boxes.

Connect to and Disconnect from a Replication Environment

The Replication Manager saves the environment information so that you do not have to re-create it when you restart Sybase Central. You only need to connect to or disconnect from an existing replication environment.

Connecting to an Existing Replication Environment

Use Replication Manager to connect to an existing replication environment.

1. Select the environment to which you want to connect.
2. In the login dialog box, enter your user name and password.
3. Click **OK**. This allows you to start managing the environment.

You can be connected to more than one environment or RMS domain at a time.

Disconnecting from a Replication Environment

Use Replication Manager to disconnect from a replication environment.

1. Select the environment from which you want to disconnect.
2. Select **Tools > Disconnect** . The Details view displays the state of the environment you disconnected from.

Setting Up a Replication Environment Using Replication Manager

Use Replication Manager to launch the Configure Replication wizard to help you quickly create the type of replication environment you want.

After you have created the replication environment, use the Configure Replication wizard to create connections, database replication definitions, and subscriptions for your replication tasks and one of several types of working replication environments, including:

- A simple warm standby environment
 - An environment consisting of one primary and multiple replicates
 - A bidirectional replication environment
1. Select the environment object you created.
 2. In the right pane, double-click **Configure Replication**. The Configure Replication wizard appears.
 3. In the Configure Replication wizard, select the type of environment you want to create:
 - Standard Replication Server warm standby environment.
 - An environment where the primary database is replicated to multiple replicate sites.
 - Bidirectional replication environment.

Configuring a Standard Warm Standby Environment

Use the Configure Replication wizard to set up a simple warm standby environment.

Prerequisites

If the active server you want to configure does not appear in the list, click **Add Server** to start the Add Server wizard. Follow the procedure to create a replication environment object.

Task

1. After selecting the type of environment to create from the Configure Replication wizard, click **Next**.
2. Select the active server and active database.
3. Select the Replication Server that will manage the database connections.
4. Select the standby server and standby database.

5. Enter the name of the logical connection.

When using an existing connection to create a warm standby logical connection, you must use the existing data server and database names of the active database for the logical connection name. See the *Replication Server Administration Guide Volume 2* for more information.

6. Enter the user name and password of the maintenance user. If the maintenance user does not exist, the wizard creates one for you. Accept the defaults, or enter your own values.
7. Select the user name and password that the RepAgent will use to connect to the Replication Server. If the RepAgent user does not exist, the wizard creates one for you. Accept the defaults, or enter your own values.
8. Select the materialization method.
9. Review the summary information about the replication environment.
10. If everything looks correct, click **Finish**. Otherwise, click **Back** to return to an earlier window and change the replication environment information. Then, return to the final wizard window and click **Finish**.

Replication Manager creates the following replication objects:

- Logical connections
- Physical connections
- Maintenance user in both active and standby Adaptive Server Enterprise servers.

See also

- *Creating a Replication Environment Object* on page 70

Configuring an Environment with One Primary and Multiple Replicates

Set up a replication environment where data is replicated from one site to many replicate sites using multisite availability (MSA).

Prerequisites

If the active server you want to configure does not appear in the list, click **Add Server** to start the Add Server wizard and add the server object. Follow the procedure to create a replication environment object.

Task

1. Select the primary server and the primary database. Click **Next**.
2. Select the Replication Server that will manage the database connections. Click **Next**.
3. Select a replicate server, then a corresponding database, and click **Add**.

The corresponding *data server.database* connection displays in the Connections list.

Repeat this step for each replicate server and replicate database pair you need in your environment. Click **Next**.

4. Enter the user name and password of the maintenance user. If the maintenance user does not exist, the wizard creates one for you. Accept the defaults, or enter your own values. Click **Next**.

All selected connections will use this maintenance user login.

5. Select the user name and password that the RepAgent will use to connect to Replication Server. If the RepAgent user does not exist, the wizard creates one for you, assigning a default name and password. Accept the defaults, or enter your own values. Click **Next**.

All connections will use this RepAgent login.

6. Specify how replicated tables will materialize:
 - **Create Subscription Without Materialization** – use this method if the primary data is already loaded at the replicate and updates are not in progress.
 - **Define Subscription for Bulk Materialization** – in this method, a subscription is initialized by a user-specified mechanism outside the replication system.
7. If you chose **Define Subscription for Bulk Materialization**, click **Use Dump Marker in the Transaction Log** to use **dump** and **load** coordination. Click **Next**.
8. Click **Finish** if everything looks correct in the information summary of replication environment. Otherwise, click **Back** to return to an earlier window and change the replication environment information. Then, return to the final wizard window and click **Finish**.

At the end of the configuration, Replication Manager creates the following replication objects:

- Physical connections
- A database replication definition for the primary database
- One or more database subscriptions for each of the replicate databases
- Maintenance user in Adaptive Server Enterprise servers

See also

- *Creating a Replication Environment Object* on page 70

Creating a Bidirectional Replication Environment

Define an environment where data is updated at multiple locations and replicated on each site.

1. Identify the servers and databases that will be part of the bidirectional replication environment.
2. Select a replicate server, then a corresponding database, and click **Add**.

The corresponding *data server.database* connection displays in the Connections list.

Repeat this step for each replicate server and replicate database pair you need in your environment. Click **Next**.

3. Enter the user name and password of the maintenance user. If the maintenance user does not exist, the wizard creates one for you. Accept the defaults, or enter your own values. Click **Next**.

All selected connections will use this maintenance user login.

4. Select the user name and password that the RepAgent will use to connect to Replication Server. If the RepAgent user does not exist, the wizard creates one for you, assigning a default name and password. Accept the defaults, or enter your own values. Click **Next**.

All connections will use this RepAgent login.

5. Specify how replicated tables will materialize:
 - **Create Subscription Without Materialization** – use this method if the primary data is already loaded at the replicate and updates are not in progress.
 - **Define Subscription for Bulk Materialization** – in this method, a subscription is initialized by a user-specified mechanism outside the replication system.
6. If you chose **Define Subscription for Bulk Materialization**, click **Use Dump Marker in the Transaction Log** to use **dump** and **load** coordination. Click **Next**.
7. Click **Finish** if everything looks correct in the information summary of replication environment. Otherwise, click **Back** to return to an earlier window and change the replication environment information. Then, return to the final wizard window and click **Finish**.

At the end of the configuration, Replication Manager creates the following replication objects:

- Physical connections
- A database replication definition for the primary database
- One or more database subscriptions for each of the replicate databases
- Maintenance user in Adaptive Server Enterprise servers

Manage Replication Server Objects

You can use Replication Manager to create and manage Replication Server objects, such as connections, replication definitions, subscriptions, and queues.

For non-Sybase data servers, Replication Manager uses DirectConnect™ to communicate with the data servers and to act as an interface for Replication Manager. The status of DirectConnect is reflected in the status of the non-Sybase data server.

Note: Replication Manager does not support database replication definition, database subscription, and creation of logical connections for non-Sybase data servers.

DirectConnect™ for Oracle is a component of ECDA for Oracle. With Replication Server Options 15.5, ExpressConnect for Oracle replaced ECDA for Oracle. Without ECDA for Oracle, Replication Manager cannot set up and monitor replication into an Oracle database.

Connections

Connections go from a database to a Replication Server, or from a Replication Server to a database. Replication Servers distribute transactions received from primary databases through connections to the replicate databases they manage.

Creating a Connection

Use Replication Manager to create a connection to a database.

1. Select the Connection folder from the Sybase Central object tree.
2. Double-click the **Add Connection** icon on the Details pane. The Add Database Connection wizard opens. Click **Next**.
3. Select an active server and database from the drop-down list. Click **Next**.
4. Enter a user name or accept the default value.
5. Enter a password. Then, click **Next**
6. Select from the given options. Click **Next**
7. Click **Finish** after checking the summary of information.
8. If you created a replicate connection through DirectConnect to a non-Sybase data server, manually execute the script that generates the tables and procedures required for replication.

Replication Definitions

A replication definition describes the source table to Replication Server, specifying the columns you want to copy.

It may also describe attributes of the destination table. Destination tables that match the specified characteristics can subscribe to the replication definition.

Replication Server provides replication at the database, table, and stored procedure levels. RM allows you to create a replication definition for a database, a table, or a stored procedure. A replication definition for a stored procedure is called a “function replication definition”. You can create, edit, and delete function replication definitions and function subscriptions.

Creating a Replication Definition on the Primary Database

Use Replication Manager to create replication definition on the primary database.

1. In the object tree, double-click the database where you want to create a replication definition. The Database Replication Definitions folder displays.
2. Double-click the Database Replication Definitions folder. The Add New Database Replication Definition window appears.
3. In the General tab, enter a replication definition name.

Note: You can specify other replication definition settings on the other given tabs.

4. Click **Replicate all DDL** if you want the DDL that is executed at the primary database to be replicated to the replicate database.
5. Click **OK**.

Subscriptions

A subscription identifies the primary database, which contains the data to be replicated.

Create a subscription on the replicate database to subscribe to a specific replication definition. You can create a subscription for any type of replication definition: databases, tables, and stored procedures.

Creating a Subscription for a Database Replication Definition

Use Replication Manager to create a subscription to a primary database.

1. In the object tree, double-click the database where you want to create a subscription.
2. Double-click the Database Subscriptions folder.
3. In the Details pane, double-click the **Add Subscription** icon.
4. Enter the name of the subscription.
5. Under Primary, select the Connection and the database replication definition that you want to subscribe to.
6. Select a materialization method from the drop-down list (optional).
7. Specify whether to Subscribe to truncate table (optional).
8. Click **OK**.

Queues

Data that is passed between servers (Adaptive Server, Replication Server, and so on) is stored in stable queues within Replication Server. Replication Manager displays the statistics of queue usage and the content of the queues.

Viewing Queue Data

Use Replication Manager to view queue data.

1. In the object tree, click the Queues folder. Queues display in the Details pane.
2. In the Details pane, right-click the queue whose data you want to view.
3. Select **View Data** from the context menu. The View Data dialog box opens.
4. To filter data shown, select one of the filter fields.
5. To sort the data, select **Segment, Transaction, Origin, Size, Status, Commit time, or User**.

View Queue Data Dialog Box

The View Queue Data dialog box lets you filter and sort the data from a queue as an aid in troubleshooting transactions in the queue. You can also edit, delete, or undelete a given command, or purge the first transaction in the queue.

The View Queue Data dialog box contains these options:

- Filter fields, which let you select the type of filters that RM uses to display data from the queue. These filters include:
 - Column
 - Column value
 - Segment
 - Number of blocks displayed
 - Number of rows displayed
 - Whether to start at the first segment
 - Whether to include all data to the end of the segment
 - Whether to include all rows
 - Whether to show deleted data
 - Whether to view all data to the end of the queue
- General buttons, which let you:
 - Display the queue data with the current filters
 - Close the dialog box
 - Purge the first transaction from the queue
 - Edit transactions
 - Delete transactions
 - Undelete transactions
 - Group transactions, which returns the Queue Data scrolling list display back to grouped transactions
- Queue Data scrolling list, which contains rows of data from the current queue. Each column contains specific information about the command and transaction contained in each row. For example, to sort the queue data by a specific column, select the column name. The Queue Data scrolling list refreshes, sorting the data according to that column. An arrow displays next to the column name to show that you have sorted the data by that column. The columns you can sort by include:
 - Segment
 - Transaction Name
 - Command
 - Origin Site
 - Origin Commit Time
 - Origin User

- Transaction ID
- Origin QID

Note: You can delete, undelete, or purge queue transactions only when Replication Server is in standalone mode.

Monitor a Replication Environment Using RMS

To monitor a complex replication environment, the Replication Manager connects to the servers in the environment through the Replication Monitoring Services—a middle-management layer that provides monitoring capabilities for replication environment.

RMS is an optional component of a replication system. RMS is used to monitor large or complex environments. RMS also provides the ability to control the flow of data and set the configuration parameters.

In a three-tier solution, you set up an RMS server to help you monitor your replication environment. In this solution, RMS monitors the health and availability of the servers and other components in your replication environment.

See the *Replication Server Configuration Guide* for your platform to start, stop and configure RMS.

Prepare for a Three-tier Solution

When you are creating a three-tier environment, you must connect to an RMS server.

In this environment, you must edit the `interfaces` file or the `sql.ini` file (Windows) with the host name, port number, and server name. You can edit this file using a text editor or the **dsedit** utility. You can use the same `interfaces` file that the other servers in the replication environment use.

RM does not need an entry in the `interfaces` file for RMS. You can provide the host name and port number of RMS directly to RM. The servers that are managed by RMS must be in the RMS `interfaces` file.

See the *Replication Server Configuration Guide* for your platform for information about configuring RMS.

Connecting to RMS

Connect to RMS using the RM plug-in once it has been started. You can also use the RM plug-in to add servers that need to be monitored by the RMS.

1. Click the **Connect** icon from the toolbar to open the Connect to a Replication Domain window.
2. Select **RMS Server**.

3. Enter the user name and password needed to connect to RMS.
4. Select **RMS** from the list of servers in the drop-down list, or click the **Options** button to provide the connection information for the RMS.
5. Enter a server name, host, and port number.
6. Click **OK**. The RMS server is added to your object tree.

See *Replication Server Reference Manual > Replication Monitoring Services API*.

Add and Drop Servers Through RMS

Servers in a three-tier environment are added and dropped in the same way as in the two-tier environment. The difference is in the properties that display if you select the object and view its properties.

RMS is designed to monitor servers and components, thus you can only view properties that RMS uses to monitor and troubleshoot the replication environment.

View Managed Objects

Use the RMS folder to view objects managed by RMS.

In the object tree, double-click or expand the RMS folder to view replication objects managed by RMS. Under RMS, you can still see the connections, routes, queues, and threads. When you select a replication object such as the Routes folder, you can view the list of created routes. You can manage these replication objects using the Replication Manager.

Viewing objects in the Replication Manager for RMS is exactly the same as viewing objects in a two-tier environment.

Add Event Triggers

You can use RMS to execute a script based on an event trigger to monitor the replication environment.

Replication Monitoring Services is designed to monitor the replication environment. When something happens in your environment, the server and component status changes. These changes are displayed in the event log. You can use RMS to create event triggers to monitor these changes.

Event triggers notify you when some event occurs in the replication environment. RMS executes a script when the specified event occurs. For example, a user can set up a script to request an e-mail message when a connection becomes suspended. This capability lets you specify a method of notification when an event occurs. You can create an event trigger for any server or component that the RMS monitors.

Creating an Event Trigger for a Replication Server

Use RMS to create an event trigger.

1. In the object tree, select the Replication Server.
2. On the right side of the desktop, select the Event Log pane.
3. Double-click the **Add Server Event Trigger** icon.
4. Select the status change that will trigger the event.
5. Enter a “Wait before executing” value (optional). This causes RMS to wait for the event to change before executing the trigger.
6. Select **Execute at each interval** (optional). This causes RMS to execute the trigger at each monitoring interval instead of just once.
7. Enter the name of the script for RMS to execute when the event occurs.
8. Click **OK**. The new event displays in the Event Log pane.

Manage a Replication System

You can start and shut down Replication Server, and monitor, maintain, and configure the replication system.

Set Up a Replication System

Learn the basic steps in setting up a replication system. This process requires planning and careful attention to the replication needs of your organization.

If you are new to Replication Server, refer first to the *Replication Server Design Guide* for information that can help you plan your replication system.

You can perform some of these steps in **rs_init**, Replication Server configuration utility, which allows you to configure Replication Servers and add databases to your system. You can use Sybase Central to perform most of the tasks listed here, including adding databases, creating replication definitions, and creating subscriptions.

Before you set up replication system, install your Sybase software according to the *Replication Server Installation Guide* for your platform.

After you install Replication Server, use the **rs_init** utility program to start and configure the replication system and to add databases.

See the *Replication Server Configuration Guide* for more information about **rs_init**.

Create Connections and Routes

To replicate data from one database into another, you must first establish the routes and connections that allow Replication Server to move the data from its source to its destination.

- Create connections

When you use Sybase Central or **rs_init** to add a database to your replication system, the program creates the connection for you. You never have to create a connection using the command line option **create connection** unless you are replicating data to or from a database that is not an Adaptive Server database.

See the *Replication Server Configuration Guide* to use **rs_init**.

- Create routes

Create routes using Sybase Central or the **create route** command at the source Replication Server.

See also

- *Manage Database Connections* on page 189
- *Manage Routes* on page 165

Set Permissions and Security

Set up login names, passwords, and permissions to establish Replication Server security for the replication system.

Replication Server login names and specific permissions are required for:

- Users who are setting up replicated data or monitoring and managing the Replication Server. You can create these users in Sybase Central or at the command line.
- Components of the replication system, such as the data server and the Replication Server. You can create system users in **rs_init** or at the command line.
See the installation and configuration guides for your platform for information about **rs_init**.

If network-based security is enabled at your site, you can set up secure pathways and choose message protection options for Replication Server to Replication Server communications.

See also

- *Manage Replication Server User Security* on page 243
- *Manage Replication Server System Security* on page 231
- *Manage Network-based Security* on page 261

Verify the Replication System

You must verify that the entire replication system is working before you create replication definitions or subscriptions or perform system diagnostics.

See *Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server > Verifying a Replication System*.

Create Replication Definitions

To set up a table for replication, mark it as replicated in Adaptive Server and define a replication definition for it in Replication Server. The replication definition describes the table and contains information about the columns to be replicated.

- If you plan to replicate stored procedures, create the stored procedure in both the primary and replicate database.
- If you are replicating the procedure from the primary to replicate database, mark the stored procedure for replication in the primary database.
- If you are replicating the procedure from the replicate to the primary database, mark the stored procedure for replication in the replicate database.

Create a function replication definition for the stored procedure at the primary Replication Server, even if you are replicating the stored procedure from a replicate database to the primary database.

See also

- *Manage Replicated Tables* on page 303
- *Manage Replicated Functions* on page 379

Create Subscriptions

If you create a replication definition for a table at Replication Server, you must create a subscription for that table replication definition at the replicate database. A subscription instructs Replication Server to copy data from primary tables to specified replicate databases.

Similarly, if you create a function replication definition for a stored procedure, you must create a subscription for that function replication definition at the replicate database. However, you do not need to create subscriptions for table or function replication definitions that update primary databases.

See also

- *Manage Subscriptions* on page 397

Perform Replication Server Tasks

Use **rs_init**, Sybase Central , or **isql** with RCL commands to interact with Replication Server.

rs_init allows you to set up a new Replication Server and add new databases to the system.

The Sybase Central Replication Manager plug-in component provides a graphical user interface for performing many of the tasks associated with managing a Replication Server system.

You execute RCL commands by connecting to Replication Server using a client application. You can use a utility program such as Sybase Central or **isql**, or you can use a custom application program that you create with Open Client Client-Library.

RCL commands are similar to Transact-SQL commands. See *Replication Server Reference Manual > Replication Server Commands* for the complete syntax for all RCL commands.

Since many of the commands are used on an as-needed basis, **isql** is a convenient way to execute them.

Use rs_init

Use **rs_init** to configure a new Replication Server and to add Adaptive Server databases to your replication system.

If you have an existing Replication Server, you can use **rs_init** to upgrade to a new version or downgrade to a previous version. **rs_init** is installed with the Sybase software. You can use it interactively or with a resource file.

See the *Replication Server Configuration Guide* for your platform for instructions to use **rs_init**.

Manage Replication Server with Sybase Central

Sybase Central is a Replication Server system management tool.

Sybase Central provides a graphical user interface that allows you to monitor the components of the replication system and perform Replication Server tasks.

With Sybase Central, you can:

- View a graphical representation of the topology of the replication system, which allows you to group objects and view status information. Sybase Central also provides menus for performing tasks and monitoring objects.
- Display multiple Replication Server connections and selectively view the contents of queues.
- Perform many of the tasks available from the Replication Server command line and **isql**, often more quickly than using equivalent Transact-SQL or RCL commands. For example, you can manage users, create routes and connections, create replication definitions and subscriptions, and manage warm standby applications.

See also

- *Manage Replication Environment with Sybase Central* on page 61

Use isql

You can use **isql** to interact with Replication Server.

You can use the **isql** utility to execute:

- ERSSD (Embedded Replication Server System Database) commands, using the primary user name and password from the Replication Server configuration file.
- RCL commands interactively
- Scripts stored in text files

For simple operations, using **isql** interactively may be easiest.

For more complex operations, Sybase recommends using **isql** to execute scripts, so you can keep a record of the RCL commands you have executed to set up a Replication Server. You can edit scripts and resubmit them whenever necessary. Scripts are also useful when you are verifying a new system or investigating the cause of a failure.

You can use **isql** to log in to Replication Server or Adaptive Server. You can use **isql** with both the interactive and script methods to interact with Replication Server. For information about using **isql** with Adaptive Server, refer to the Adaptive Server utility programs manual for your operating system.

Using isql Interactively

You can use **isql** interactively for simple operations.

1. If necessary, start the Replication Server.
2. Log in to the Replication Server using the following command:

```
isql -User_name -Ppassword -Sserver_name
```

Specify the name of the Replication Server using the **-S** flag.

If your login is accepted, **isql** displays a prompt:

```
1>
```

3. Enter the RCL command you want to execute.

When you press the Return key at the end of a line, **isql** increases the line number. Some commands require more than one line.

4. To execute the command, enter “go” (on a line by itself, with no blanks) and press Return.

To cancel the command, enter “reset” and press Return. The prompt’s line number is reset to 1.

Some RCL commands display immediate results. Others execute asynchronously, that is, they return a system prompt without necessarily having completed the desired action and report only syntax errors.

5. To exit **isql**, enter “quit” at the beginning of a line.

Note: You can check the status of asynchronous commands by executing RCL commands that display status or by querying the RSSD system tables at the affected sites. See *Replication Server Reference Manual > Replication Server System Tables* for more information on system tables and the stored procedures you can use to query them.

See also

- *Starting Replication Server* on page 88

Using isql to Execute Scripts

You can create scripts of RCL commands and execute the scripts using **isql**, which is useful when you need to execute the same set of commands in Replication Servers at multiple sites.

1. Start Replication Server if it is not running.
2. Create a text file for your script, and enter into it the RCL commands you want to execute. As with the interactive method, separate each command with a **go** command on a line by itself.
3. Execute the script using this **isql** syntax:

```
isql -User_name -Ppassword -Sserver_name
      -iscript_name
```

isql displays the results from the commands in the script on your screen as the standard output. Optionally, you can redirect the output to a file:

Manage a Replication System

```
isql -User_name -Ppassword -Sserver_name  
      -iscript_name > output_file
```

See also

- *Starting Replication Server* on page 88

Starting Replication Server

Start Replication Server and replication system components in sequence.

Normally, you need to restart Replication Server only if you are reconfiguring system files or if your system experienced a failure that brought down Replication Server. Initially, the installation process starts the replication system for you.

1. Start the data server containing the databases that Replication Server manages.
2. If Replication Server uses Adaptive Server Enterprise for the RSSD, start the RSSD. For more details, see the *Replication Server Installation Guide* for your platform.
3. Start Replication Server by running the **repserver** on UNIX systems or **repsrvr.exe** on Windows systems, or by executing the Replication Server run file.
4. Start RepAgent for the data server and for the RSSD if RepAgent has not been configured to start automatically at server startup.
5. To ensure that Replication Server started with no errors:
 - a) Check the `repserver.log` file for error messages (indicated with the letter “E” on the left), as described in *Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions > Error log files > Replication Server Error Log*.
 - b) Use **isql** to log in to each Replication Server, or use a script that logs in to each server. See *Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server > Monitoring Replication Server > Verify Server Status*.

See also

- *Replication Server Executable Program* on page 88

Replication Server Executable Program

Use **repserver** or **repsrvr.exe** at the operating system prompt to run the Replication Server program.

For example, to run **repserver**, log in to the operating system as the “sybase” user, and execute **repserver** using the following syntax:

```
repserver [-C config_file] [-i id_server] [-S rs_name]  
          [-I interfaces_file] [-E errorlog_file] [-M] [-v]  
          [-K keytab_file]
```

See *Replication Server Reference Manual > Executable Programs* for complete information about each of the parameters of the **repserver** command.

The **rs_init** program creates the run file “RUN_*name*,” where *name* is the name of the Replication Server. The run file specifies the **repserver** command with parameters set for the installed Replication Server. Normally, you start Replication Server by executing the run file.

The Replication Server executable program is located in the `bin` subdirectory, and the Replication Server run file is in the `install` subdirectory of the Sybase release directory. Refer to the Replication Server installation and configuration guides for your platform for more information.

Replication Server Configuration File

Replication Server finds the startup information it needs in the Replication Server configuration file.

The file is created by the **rs_init** program, but it can be edited with a text editor. If it contains encrypted passwords, however, you must modify them using **rs_init**. See the Replication Server installation and configuration guides for your platform. The default name for the Replication Server configuration file is the Replication Server name with “.*cfg*” appended.

Stopping a Replication Server Using isql

Stop a Replication Server using the **isql** utility.

When you shut down a Replication Server, it refuses additional connections, terminates threads, and exits.

1. Use **isql** to log in to the Replication Server as the System Administrator:

```
isql -Usa -Psa_password -Sservername
```

2. Enter:

```
shutdown
go
```

Add a Replication Server

To add a Replication Server to a replication system, use the **rs_init** program, as described in the installation and configuration guides for your platform.

Always conduct a careful review and analysis of how the additional Replication Server will fit into your system. Determine the other processes that are required for the server and designate required names and accounts for these processes.

The first Replication Server you install must be the ID Server. It must be running when you install new Replication Servers or add databases to the replication system.

Manage a Replication System

You can use Sybase Central to add Replication Server to a replication system when you are creating a replication environment.

When you install each Replication Server, **rs_init** performs the following tasks:

- Creates a configuration file for the Replication Server
- Creates an executable run file to start the Replication Server
- Sets RepAgent parameters at Adaptive Server
- Creates and initializes the RSSD or the ERSSD.
- Starts the Replication Server and RepAgent for the RSSD, as necessary

After you have executed **rs_init** for each Replication Server you are adding:

1. Determine the routing for the Replication Server, and modify the routes in the existing system to accommodate the new Replication Server.
2. If you want to add a new database, prepare that database for replication.
3. Grant users the appropriate permissions for Replication Server commands.
4. If applicable, add replication definitions, subscriptions, function-string classes, and error classes for the Replication Server.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations* and *Replication Server Administration Guide Volume 2 > Errors and Exceptions Handling*.

See also

- *Manage Routes* on page 165
- *Manage Database Connections* on page 189
- *Manage Replication Server Security* on page 231
- *Manage Replicated Tables* on page 303
- *Manage Subscriptions* on page 397
- *Creating a Replication Environment Object* on page 70

Add a Replication System Domain

A replication system domain includes all replication system components that use the same ID Server.

Most replication systems should be set up as a single domain with a single ID Server. However, you may require replicates of two separate data environments in the following situations if:

- Your enterprise requires data management by separate groups, sites, or independent organizations.
- You need to eliminate an ID Server as a single point of failure, thereby creating a fault-tolerant system.

An ID Server failure in a domain results in system degradation. New Replication Servers and databases cannot be added to a domain as long as the ID Server is shut down.

If you do use multiple replication system domains, be sure to have completely independent data environments. For example, assume you have one data environment tracking personnel, and another tracking inventory. As long as there is no data sharing or relationship between these two groups, you can create two separate domains, one for each data environment.

Guidelines for Adding Replication System Domains

You must observe certain guidelines before you create multiple ID Servers for multiple replication system domains.

- Make sure all Replication Server and data server names are globally unique across domains.
By using unique names, you simplify your administration and prevent confusion, especially in the interfaces files, which contain network access information for servers.
- You need to assign the ID number each time a Replication Server or database is added to the replication system.
- Maintain unique names and distinct ID numbers to accommodate the future possibility of data transfer between domains (that is, merging of domains).
 - Provide a different range of database and Replication Server ID numbers for each domain.
 - Make sure the ID numbers of any additional domains are large enough so that they do not overlap with the ranges of the first domain.
- Make sure that replication definition names are globally unique within and between ID Server domains.

Examples of Assigning ID Numbers

You can only use the ID numbers within specified ranges for each ID Server in your replication environment.

The ID number is increased each time a Replication Server or database is added to the replication system. By default, your first ID number for a data server is 101. For a Replication Server it is 16,777,317. The last possible ID number for a data server is 16,777,316. For a Replication Server it is 33,554,431.

If you are creating two domains, you could assign ID numbers according to the table.

Table 3. Suggested ID Numbers for Multiple ID Servers

Component	First ID number	Last ID number
1st domain data server	101	99,999
2nd domain data server	100,000	16,777,316
1st domain Replication Server	16,777,317	17,777,316

Component	First ID number	Last ID number
2nd domain Replication Server	17,777,317	33,554,431

When you are installing an ID Server using the **rs_init** program, you can specify the Starting Replication Server ID and the Starting Database ID.

Note: Make sure your ranges do not overlap from one domain to another. Make your ranges large enough so that ID numbers can never increase to the next range. For example, a range of 99,999 accommodates nearly all possible cases.

Set Replication Server Configuration Parameters

You can configure Replication Server or specific objects within the replication system by using one of several methods that update configuration parameters in the `rs_config` system table in the RSSD or the ERSSD.

You can also check configuration status information in this table.

Note: Replication Server startup information is stored in a configuration file created by **rs_init**. The default name for the Replication Server configuration file is the Replication Server name with “.cfg” appended. See *Replication Server Reference Manual > Executable Programs* for more information about the configuration parameters stored in this file.

Replication Server Configuration Parameters

Replication Server reads configuration parameters from the `rs_config` system table in the RSSD or ERSSD.

All configuration parameters have default values, which are inserted in the table when you use the **rs_init** utility to create the RSSD or ERSSD. The default values are sufficient for most replication systems. Normally, you change default values only for unusual environments or special situations. For example, you may need to adjust parameters for performance tuning if your system has a large number of replication definitions or subscriptions. You can change default values using **configure replication server**.

Basic parameters governing the names and version numbers of the Replication Server and its components can also be set with **rs_init**.

rs_init also sets the password encryption configuration parameter.

Many configuration parameters also have values for specific objects. You set these values after installation when your system requires the fine-tuning that these parameters allow. For example, default route parameters affect all routes that originate at the current Replication Server. If necessary, you change the default settings for these parameters with **configure replication server**. You also can set parameter values for individual routes by using the **alter route** command.

Setting some configuration parameters requires a technical understanding of the replication system.

It is important to back up the RSSD periodically, and whenever you do anything to change its state. ERSSD is already configured for daily automated backup.

See *Replication Server Administration Guide Volume 2 > Performance Tuning*. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance*.

See also

- *Replication Server Technical Overview* on page 23
- *Manage the RSSD* on page 96
- *Manage the Embedded Replication Server System Database* on page 98

Basic Configuration Parameters

Basic parameters govern the names and version numbers of the Replication Server and its components.

Warning! Do not change the values for the basic configuration parameters. The values are set when you run **rs_init** and should only be modified by the **rs_init** program when you upgrade or downgrade Replication Server.

Table 4. Basic Configuration Parameters

Configura- tion Param- eter	Description
current_rssd_version	The Replication Server version supported by this RSSD. The Replication Server checks this value at startup.
id_server	The name of the ID Server for this Replication Server.
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.
oserver	The name of the current Replication Server.
prev_min_rssd_version	Following an rs_init installation upgrade, this value contains the previous value of minimum_rssd_version .
prev_rssd_version	Following an rs_init installation upgrade, this value contains the previous value of current_rssd_version .
rssd_error_class	Error class for the RSSD. Default: rs_sqlserver_error_class

Configura- tion Param- eter	Description
send_enc_pw	Ensures that Replication Server makes client connections to the RSSD with an encrypted password. Values are “on” and “off” (the default).

See also

- *Send Encrypted Passwords for Replication Server Client Connections* on page 240

Different Types of Configuration Parameters

There are different types of configuration parameters in the `rs_config` system table that affect the Replication Server and different database objects.

The method for changing a parameter also varies according to the object that the parameter affects. The different types of configuration parameters are:

- Local Replication Server – parameters whose effects are restricted to the current Replication Server. These include the basic configuration parameters and the parameters listed in *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance > Replication Server Parameters that Affect Performance*.
- Route – parameters that affect routes from the current Replication Server to other Replication Servers.
- Database connection – parameters that affect database connections originating with the Replication Server. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads > Parallel DSI parameters*.
- Logical database connection – Replication Server parameters that apply to logical database connections for warm standby applications. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Alter Warm Standby Database Connections > Alter Logical connections > Change Parameters Affecting Logical Connections* for information about setting default and per-target values for logical connection parameters.
- Network-based security services – parameters that affect network security.
- Performance – parameters that affect the performance of a Replication Server. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance* and *Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads > Parallel DSI parameters*.

See also

- *Change Replication Server Parameters* on page 95
- *Change Routes* on page 177
- *Set and Change Parameters Affecting Physical Connections* on page 198

- *Manage Network-based Security* on page 261
- *Basic Configuration Parameters* on page 93

Change Replication Server Parameters

You can modify configuration parameters that affect the current Replication Server by using **configure replication server** at the Replication Server.

To change default configuration parameter values using **configure replication server**, log in to Replication Server and execute **configure replication server** at the **isql** prompt.

Use this syntax where *config_param* is a character string that corresponds to the configuration parameter name and *value* is a character string representing the setting you want for the parameter:

```
configure replication server
    set config_param to 'value'
```

The *config_param* string must match an entire parameter name. You may have to restart Replication Server for the new parameters to take effect.

Example 1

For example, to change the maximum number of messages allowed in the Open Server message queue to 5, log in to the source Replication Server and:

1. Execute the **configure replication server** command:

```
configure replication server set num_msgs to '5'
```

2. Restart the Replication Server.

Example 2

This example uses **configure Replication Server** to change the **ha_failover** parameter to enable Failover support for all non-RSSD connections from a Replication Server to Adaptive Servers.

1. Execute **configure replication server**. Log in to the Replication Server for which you want to enable Failover support and enter:

```
configure replication server
set ha_failover to 'on'
```

See *Replication Server Administration Guide Volume 2 > Replication System Recovery > Configuring the Replication System to Support Sybase Failover*.

2. Restart the Replication Server.

Configuration changes take effect after you restart Replication Server.

See *Replication Server Reference Manual > Replication Server Commands > configure replication server*.

Be aware of parameters affecting security and see *Replication Server Administration Guide Volume 2 > Performance Tuning* for parameters affecting performance.

See also

- *Manage Replication Server Security* on page 231
- *Starting Replication Server* on page 88

Configure Dynamic Parameters

For dynamic configuration parameters, you do not need to restart the Replication Server for the new values to take effect. Use **configure replication server** to modify the parameter value.

Use **admin config** to retrieve the values of these parameters.

The **admin config** syntax is:

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

See the *Replication Server Reference Manual* for a detailed information in using this command.

Dynamic Configuration Parameters

Use **configure replication server** with dynamic configuration parameters to affect Replication Server behaviour without having to shut down and restart Replication Server.

Table 5. Dynamic Configuration Parameters

init_sqm_write_delay	init_sqm_write_max_delay
memory_limit	num_concurrent_subs
queue_dump_buffer_size	sqm_recover_segs
sqm_warning_thr_ind	sqm_warning_thr1
sqm_warning_thr2	sqt_max_cache_size
sqt_init_read_delay	sqt_max_read_delay
sts_cachesize	sts_full_cache_system_table_name

Manage the RSSD

The data in each Replication Server RSSD is essential in keeping the replication system running.

The replication system administrator or Adaptive Server system administrator manages the RSSD by monitoring the condition of the database and performing regular dumps. In the event of disaster recovery, you need to rely on up-to-date backups of the RSSD for full system recovery. Therefore, it is critical that you perform periodic backups of the replication system.

It is also important to back up the RSSD after performing tasks that change its state, such as adding routes, replication definitions, and subscriptions, or altering function strings for databases to which you are connected.

The system tables are loaded into the RSSD during Replication Server installation. You can query the system tables to find the status of the system, but in general, you should not make changes to the tables directly. Refer to the *Replication Server Reference Manual* for detailed descriptions of the system tables.

Enable Failover Support for an RSSD Connection

Use Sybase Failover to configure two version 12.0 or later Adaptive Servers as companions. If the primary companion fails, the devices, databases, and connections of the primary companion can be taken over by the secondary companion.

For more detailed information about how Sybase Failover works in Adaptive Server, see *Using Sybase Failover in a High Availability System* which is part of the Adaptive Server Enterprise documentation set.

To enable Failover support for an RSSD connection, use either of the following methods:

- Use **rs_init** when you install a new Replication Server.
For instructions, see *Replication Server Configuration Guide > Configure Replication Server and Add Databases Using rs_init* for your platform.
- Edit the Replication Server configuration file after you have installed the Replication Server.

1. Use a text editor to open the Replication Server configuration file. The default file name is the Replication Server name with a “.cfg” extension.

The configuration file contains one line per entry.

2. Find the line “RSSD_ha_failover=no” and change it to:

```
RSSD_ha_failover=yes
```

3. To disable Failover support for an RSSD connection, change the “RSSD_ha_failover=yes” to:

```
RSSD_ha_failover=no
```

These changes take affect immediately; that is, you do not have to restart Replication Server to enable Failover support.

If you cannot recover the most recent database state of the RSSD, see *Replication Server Administration Guide Volume 2 > Replication System Recovery > Recover from RSSD failure*.

Note: It is not possible to migrate an RSSD database across platforms using commands such as, cross-platform **dump** and **load**, or **bcp**. To migrate, you must rebuild the replication system on the new platform.

For instructions on how to enable Failover support for non-RSSD Replication Server connections to Adaptive Server, see *Replication Server Administration Guide Volume 2 >*

Replication System Recovery > Configure the Replication System to Support Sybase Failover.

Manage the Embedded Replication Server System Database

Replication Server can run on either an Adaptive Server Enterprise Replication Server System Database (RSSD) or on an Embedded RSSD (ERSSD). ERSSDs are designed for users who do not want to use Adaptive Server Enterprise to manage the Replication Server RSSD.

Replication Server is easy to install and manage with ERSSD. ERSSD is automatically installed, configured, and started in the background if you specify that you want to use it. Backup procedures are automatic and preconfigured.

Remember: You cannot migrate from ERSSD to RSSD. To use ERSSD, you must select it when you install Replication Server. See the *Replication Server Installation Guide*.

Sybase provides ERSSD as an option in Replication Server, implemented in SQL Anywhere. Sybase continues to support the traditional RSSD, implemented in Adaptive Server Enterprise. All the ERSSD features discussed pertain to ERSSD only; they do not affect the behavior of the traditional RSSD in Adaptive Server Enterprise.

ERSSD runs on three operating system files:

- A database root file
- A transaction log file
- A transaction log mirror file

When you start **rs_init**, provide the directories for these files, and make sure that the name of your ERSSD is in the interfaces file.

Tip: For better performance and better protection against disk failure, put each one of these files on a different physical device.

Obtain Information on ERSSD Settings

Learn how to obtain configuration settings and other information on ERSSD.

ERSSD has a preconfigured backup time, backup interval, and backup directory. Unless you want to change these defaults, you need not configure ERSSD. There are four files in the backup directory. This directory is specified when you install Replication Server with ERSSD. To check the current default values, enter:

```
sysadmin erssid
```

In the Replication Server configuration file, you can find the:

- ERSSD database file path

- ERSSD transaction log file path
- ERSSD transaction log mirror file path
- Backup directory path

ERSSD Configuration Parameters and Command

Use the ERSSD configuration parameters to configure backup time and backup directory, and ERSSD routing.

Syntax

```
configure replication server
set
    {erssd_backup_start_time |
    erssid_backup_start_date |
    erssid_backup_dir |
    erssid_backup_interval | erssid_ra}
to 'value'
```

Warning! Do not update these values directly in the `rs_config` table.

Table 6. ERSSD Configuration Parameters

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.	01:00 AM
erssd_backup_start_date	Date the backup begins. Specified as “MM/DD/YYYY”	Current date
erssd_backup_interval	Interval between backups of database and log. Specified as “nn hours” or “nn minutes” or “nn seconds”.	24 hours
erssd_backup_dir	Location of stored backup files. Should be a full directory path. Configuring this path causes an immediate, unscheduled backup.	Same directory as the transaction log mirror; initial value specified in rs_init
erssd_ra	Its value should be a server name. It is only used when when a user creates a route from the current site.	<i>erssd_name_ra</i> where <i>erssd_name</i> is the ERSSD name in the your replication system

Backup ERSSD

An automatic full backup, including both the database file and the transactional log file, is performed at the default or the configured time. You can perform an unscheduled backup for ERSSD.

There are four files in the backup directory. This directory is specified when you install Replication Server with ERSSD.

The transaction log is mirrored, providing extra protection for critical data, and enables complete recovery of the transaction log file.

To perform an unscheduled backup, enter:

```
sysadmin erssid, backup
```

ERSSD Backup Directory Files

The Replication Server backup directory contains backup files for the ERSSD database and transaction log. The backup directory is specified when you install Replication Server with ERSSD.

Table 7. Backup Directory Files for ERSSD

File name	File definition
<i>erssd_name.db</i>	Backup database file
<i>erssd_name.log</i>	Backup transaction log
<i>erssd_name.db.pre</i>	Previous backup database file
<i>erssd_name.log.pre</i>	Previous backup transaction log

ERSSD Routing

Use **create route** to create a route from Replication Server with ERSSD.

You can create a route from a Replication Server with ERSSD, as long as both the source and the destination servers are version 15.0 or later. Verify that the Replication Agent name is in the Replication Server interfaces file; an ERSSD Replication Agent is started as an open server during **create route** and if its name does not appear in the interfaces file, the command fails

The default ERSSD Replication Agent name is *erssd_name_ra*. To replace the default name with that of your Replication Agent server, enter:

```
configure replication server
set erssid_ra to 'value'
```

Note: Sybase provides ERSSD in SQL Anywhere (SA) as an option, and continues to support the traditional RSSD in the Adaptive Server Enterprise.

Move ERSSD Files

Use **sysadmin erssd** to move the ERSSD database file, transaction log, or transaction log mirror.

Do not edit the configuration file itself. Moving the database file, transaction log, and transaction log mirror is an expensive operation. Only use it when you are sure it is necessary. For more information on **sysadmin erssd**, see the *Replication Server Reference Manual*.

ERSSD User Administration

You can change ERSSD user passwords, add users, and assign permissions.

There are only two users in the ERSSD, the primary user, who also acts as system administrator, and the maintenance user. You can find their names and passwords in the configuration file. You can do the following to change the user password:

- Use **alter user** to alter the primary user password.
- Use **alter connection** to alter the maintenance user password.

Both these commands alter the password at Replication Server as well as at ERSSD, and update the Replication Server configuration file.

For more information on these commands see the *Replication Server Reference Manual*.

To add a user at the ERSSD, use **isql** to access the ERSSD as the primary user and execute **grant connect to *username* identified by *password***.

To give a user permission to read the Replication Server system tables, execute the command **grant membership in group rs_systabgroup to *username***.

To grant **sa** privileges to a user, execute **grant dba to *username***.

Reduce ERSSD File Size

Use **sysadmin erssd** with the **defrag** parameter to recover any unused space, shrink the ERSSD size, and rebuild the ERSSD database file.

When Replication Server deletes rows from the ERSSD, for example when Replication Server deletes exceptions from the exception log, the ERSSD database does not release the space back to the ERSSD filesystem. Therefore, you must defragment the ERSSD database to recover disk space.

The **defrag** parameter uses the **dbunload SQL Anywhere** command to rebuild the ERSSD.

Before defragmenting the ERSSD:

- Verify you have sufficient disk space in your ERSSD as defragmentation requires as much space as the original ERSSD for working space.
- Set your site version to 15.0 or later to allow you to use the **defrag** parameter.

Manage a Replication System

- Verify the replication system is not busy.

Executing **sysadmin erssd, defrag**:

1. Shuts down eRSSD.
2. Shuts down **dbltm**, the ERSSD Replication Agent, if **dbltm** is running.
3. Calls **dbunload** to rebuild and defragment the ERSSD database file. **dbunload** saves the old transaction log as `erssdName.olg` in the transaction log directory.
4. Restarts ERSSD.

The dREC recovery daemon restarts the **dbltm** for the ERSSD.

Note: During the defragmentation process, more files may be generated in the transaction log directory. Do not delete these files. Replication Server and **dbltm** will remove the files when the files are not needed.

ERSSD Recovery Procedures

ERSSD automatically manages recovery from operating system crashes, database server crashes, and crashes caused by shutting down improperly. You can use several procedures designed to recover a database damaged by media failure and ensure a clean recovery after media failure.

See the Replication Server Reference Manual > Replication Server Commands > **sysadmin erssd**.

Prerequisites for Replication Server Recovery

Before using the recovery commands, set the relevant environment variables.

On UNIX platforms:

- Set your environment variable **PATH** to include `$$SYBASE/$$SYBASE_REP/ASA11/bin`:

```
setenv PATH $$SYBASE/$$SYBASE_REP/ASA11/bin:$PATH
```

- Set your environment variable **LD_LIBRARY_PATH** (**SHLIB_PATH** on HP, **LIB_PATH** on AIX) to include `$$SYBASE/$$SYBASE_REP/ASA11/lib`:

```
setenv LD_LIBRARY_PATH $$SYBASE/$$SYBASE_REP/ASA11/lib:
$LD_LIBRARY_PATH
```

On Windows:

- Set your environment variable **PATH** to include `%SYBASE%\%SYBASE_REP%\ASA11\win32`:

```
set PATH=%SYBASE%\%SYBASE_REP%\ASA11\win32;%PATH%
```


Recovering After Media Failure of the Database File

You can recover the ERSSD database file.

1. Make an extra backup copy of the current transaction log. If the database file is gone, the only record of changes since the last backup is in the transaction log.
2. Create a recovery directory to hold the files you use during the recovery process.
3. Copy the database file from the last full backup to the recovery directory. You can find the database file in the backup directory. It is named *erssd_name*.db.
4. Copy the backup transaction log into the recovery directory. The backup transaction log, named *erssd_name*.log, is in the backup directory.
5. Apply the transactions from the backup transaction log to the recovery database:

```
dbsrv11 erssd_name.db -a erssd_name.log
```

6. Copy the online transaction log into the recovery directory. The online transaction log, named *erssd_name*.log, is in the log directory.
7. Apply the transactions from the online transaction log to the recovery database:

```
dbsrv11 erssd_name.db -a erssd_name.log
```

8. Make a post-recovery backup by making an extra copy of the database file.
9. Move the database file to the production directory and restart the database. Use the command **dbspawn** from the Replication Server error log.
10. Perform validity checks on the recovery database:

```
dbvalid -c
"uid=primary_user_name;
pwd=primary_user_password;eng=erssd_name
LINKS=tcPIP
(DOBROAD=NONE;HOST=localhost;PORT=port)"
```

11. Restart Replication Server.

Recovering from Media Failure on the Database Transaction Log

You can recover the ERSSD database transaction log.

1. Identify the corrupted file. You can do this by running the Log Translation utility on both the transaction log and its mirror to see which one generates an error message. In this example, the Log Translation utility, **dbtran**, translates a transaction log named *erssd_name*.log, saving the translated output in *db_name*.sql.

```
dbtran erssd_name.log
```

The Log Translation utility translates the intact file with no errors, but reports an error when translating the corrupt file.

2. Copy the correct file over the corrupted file, so that the two files are identical.
3. Restart the database, using **command** from the Replication Server error log.

4. Restart Replication Server.

Resetting the Database Transaction Log for ERSSD

If Replication Agent is running and there are existing routes, you can reset the database transaction log for the ERSSD after you recover from media failure on the log.

ERSSD is an SQL Anywhere database and you must execute the **dblog**, **dbeng11**, and **dbstop** SQL Anywhere commands at the system prompt. See the SQL Anywhere documentation for the command syntax and to set the environment for the commands.

1. Shut down Replication Server using **shut down**.

2. Reset the ERSSD database log at a system prompt:

```
dblog -il erssid_name.db
```

3. Move the current database log file from the `translog` directory in Replication Server to another location.

Retain the current log file for a few days in case you encounter problems with the new log file.

4. Start a new database log file.

For example, in UNIX enter:

```
dblog -t erssid_name.db $SYBASE/REP_15-5/ASA12/translog/  
erssid_name.log
```

5. Start the ERSSD database server at the system prompt:

```
dbeng11 erssid_name.db
```

6. Reset the ERSSD database locator value to 0:

a) Start **isql**.

b) Execute **rs_zeroltm** on the ERSSD database:

```
rs_zeroltm erssid_name, erssid_name
```

c) Exit **isql**.

7. Stop the ERSSD database at the system prompt:

```
dbstop -c  
"eng=erssid_name;uid=primary_user_name;  
pwd=primary_user_password"
```

8. Start Replication Server.

See also

- *Starting Replication Server* on page 88
- *Stopping a Replication Server Using isql* on page 89

Quiesce Replication Server

To quiesce a replication system means to put the system in a state in which no Replication Servers have messages to send or receive.

You may need to quiesce all Replication Servers in the system to recover databases, alter routes, and troubleshoot the system. A Replication Server is quiescent when the following conditions are true:

- Subscription materialization queues do not exist.
- Replication Server has read all messages in all queues.
- Transaction caches for inbound queues contain no complete transactions.
- Messages in RSI queues have been sent and acknowledged.
- Messages in DSI queues have been applied and acknowledged.

Quiescing a Replication System

You can use a sequence of commands or Sybase Central to quiesce a system consisting of several Replication Servers.

1. Execute the **suspend log transfer from all** command at each Replication Server. This prevents RepAgent from connecting to the Replication Servers.
2. Execute **admin quiesce_force_rsi** at each Replication Server.

This command forces Replication Server to deliver all queued messages to other Replication Servers, then reports whether the system is successfully quiesced.

Quiescing occurs most efficiently if you follow the flow of the data. For example, if data flows from TOKYO_RS to MANILA_RS to SYDNEY_RS, quiesce the Replication Servers in that order.

3. Check that the Replication Server is quiescent using **admin quiesce_check**. If necessary, repeat steps 2 and 3 until all Replication Servers are quiescent.
4. After all Replication Servers are quiescent, execute **admin quiesce_force_rsi** once more at each Replication Server. Check that each Replication Server is quiescent using **admin quiesce_check**. If necessary, repeat this step until all Replication Servers are quiescent.

This step is necessary because, although a Replication Server may be quiescent, it may have recently sent messages to another Replication Server. These messages may initiate more communication between these two Replication Servers or between several Replication Servers in the replication system. Repeating steps 2 and 3 ensures that you have quiesced the entire replication system.

Remove a Replication Server

How you remove a Replication Server from a replication system depends on whether or not the Replication Server is active (running). It is easiest to remove a Replication Server that is active. You must also drop routes and subscriptions when you remove a Replication Server.

See also

- *Manage Subscriptions* on page 397
- *Manage Routes* on page 165

Removing an Active Replication Server

Drop subscriptions, replication definitions, connections, routes, and perform other tasks in the correct sequence to ensure you can safely remove an active Replication Server from service.

1. Query the RSSD to determine what replication definitions are defined at the primary Replication Server (the server you are removing from service). You can use the **rs_helprep** stored procedure to do this. See *Replication Server Reference Manual > RSSD Stored Procedures > rs_helprep* and, see *Replication Server Reference Manual > Replication Server System Tables*
2. Drop subscriptions and replication definitions.
 - a) For each replication definition defined at the primary Replication Server, execute the **drop subscription** command for each subscription on all Replication Servers that manage subscribing data.

To retain data at the replicate, execute the **drop subscription** command **without purge**.

To delete data at the replicate, execute the **drop subscription** command **with purge**.
 - b) Drop all replication definitions for primary data managed by the Replication Server (determined in step 1).

Wait for the replication definitions to disappear from the RSSDs of Replication Servers that the Replication Server has a route to.
 - c) At the Replication Server you are removing, drop all subscriptions to replication definitions on other Replication Servers.

To retain data at the replicate, execute the **drop subscription** command **without purge**.

To purge data at the replicate, execute the **drop subscription** command **with purge**.
3. If the Replication Server is the primary Replication Server for a function-string class or error class, execute the **move primary** command at another Replication Server to change the primary Replication Server for each class.

During a **move primary** operation, routes must exist from the old primary site to the new primary site, and from the new primary site to the old primary site. The Replication Server assuming the role of the primary site also must have routes to all of the same Replication Servers as the old primary site.

4. Drop database connections.
 - a) Stop all RepAgents connected to the Replication Server, using the **sp_stop_rep_agent** system procedure at Adaptive Server.
 - b) Remove connections to all databases managed by this Replication Server, using the **drop connection** command.

Note: If you want to continue to maintain the replicate data in databases previously managed by a Replication Server that has been removed from service, you must create connections to those databases from some other Replication Server and create new subscriptions.

5. Perform the following routing tasks:
 - a) If the Replication Server is an intermediate site in a route, use the **alter route** command so it is no longer an intermediate site.
 - b) Drop all routes *from* the Replication Server.

To do this, execute the **drop route** command for each route from the Replication Server to another Replication Server.
 - c) Drop all routes *to* the Replication Server.

To do this, execute the **drop route** command on each Replication Server that has a route to the Replication Server you are removing.
6. After all subscriptions and routes to and from the Replication Server are dropped, remove the Replication Server from the list maintained by the ID Server. To do this, execute the **sysadmin droprs** command on the ID Server:

```
sysadmin droprs, replication_server
```

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

7. Remove all databases managed by the Replication Server from the database list maintained by the ID Server. Include the RSSD. To remove databases, run the **sysadmin dropdb** command on the ID Server, for each database:

```
sysadmin dropdb, data_server, database
```

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

See also

- *drop subscription Command* on page 426
- *Manage Routes* on page 165

Removing an Inactive Replication Server

An inactive Replication Server is one that is not running. To take an inactive Replication Server out of service, you must drop and purge routes, and perform other tasks to safely remove the Replication Server from service.

1. Drop all routes to the Replication Server.

To do this, execute the **drop route** command with the **with nowait** option on each Replication Server that has a route to the Replication Server. For example:

```
drop route to OLD_RS with nowait
```

This command also deletes information about subscriptions created at OLD_RS for data managed by this Replication Server.

2. If the Replication Server you are removing is primary for any function-string classes or error classes other than the system defaults, **rs_default_function_class** and **rs_sqlserver_error_class**, create a replacement for each class at a new primary. To do this:

- a) Choose a Replication Server that has routes to all other Replication Servers that use the class.
- b) Create a new class at that Replication Server containing the same function strings or error actions as the original class. See *Replication Server Administration Guide Volume 2 > Customize Database Operations* and *Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions*.
- c) Alter each database connection that is using the original class to use the new class instead.

3. On each Replication Server that has a route from the Replication Server, purge the Replication Server route.

To purge a route, execute the **sysadmin purge_route_at_replicate** command on each Replication Server to which the Replication Server had a route. For example:

```
sysadmin purge_route_at_replicate, OLD_RS
```

This command also removes:

- Subscription information for data originating at the Replication Server you are removing from service.
- Function-string and error classes defined at the Replication Server you are removing from service. If the Replication Server is the primary site for **rs_default_function_class**, **rs_sqlserver_function_class**, or **rs_sqlserver_error_class**, these classes are not removed but are reset to have no primary Replication Server.

4. Remove the Replication Server from the list maintained by the ID Server. To do this, execute the **sysadmin droprs** command on the ID Server:

```
sysadmin droprs, replication_server
```

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

5. Remove all databases managed by the Replication Server from the database list maintained by the ID Server. Include the RSSD. To remove databases, run the **sysadmin dropdb** command on the ID Server, for each database:

```
sysadmin dropdb, data_server, database
```

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

This completes the removal of an inactive Replication Server from a replication system.

Next

Keep in mind these three additional points:

- If you want to continue to replicate any data in the databases previously managed by the Replication Server, you must reassign those databases to some other Replication Server.
- Since the subscriptions to the Replication Server data did not go through normal subscription dematerialization, replicate data has not been deleted from replicate Replication Servers.
- You may need to create additional routes to maintain the replication system—for example, if the Replication Server is an intermediate on an indirect route.

See also

- *Manage Database Connections* on page 189

Manage RepAgent and Support Adaptive Server

Learn about Replication Server support for Adaptive Server features, and how to set up, configure, and manage RepAgent, the Replication Agent for Adaptive Server.

RepAgent is an Adaptive Server thread; it scans the database transaction log and sends transaction information to the Replication Server for distribution to subscribing databases.

See *Replication Server Administration Guide Volume 2 > Performance Tuning > Replication Server Internal Processing > Processes in the Primary Replication Server > Replication Agent User Thread*.

See also

- *Replication Server Technical Overview* on page 23

Set up RepAgent

After Replication Server and Adaptive Server are installed on your system, you must enable a RepAgent for each database the Replication Server manage.

Enable a RepAgent for each database the Replication Server manages if the database:

- Contains primary data, or
- Contains stored procedures marked for replication

In addition, if Replication Server is the source site for any route, you must enable RepAgent for the Replication Server RSSD.

In some scenarios for setting up RepAgent, you use **rs_init**, in other scenarios you must use command line options:

- If you install a new Replication Server or add a new database, use **rs_init** to set up RepAgent. This process enables RepAgent, set default parameters, and start RepAgent. See the *Replication Server Configuration Guide* for your platform for information about **rs_init**.
- To change an existing replicate database to a primary database, you must use command line options.

Multithreaded RepAgent

By default, the Adaptive Server RepAgent consists of a single thread that scans the primary database log, generates LTL, and sends the LTL to Replication Server. With multithreaded RepAgent, the scanning and sending activities are performed by separate threads. You can

then configure multithreaded RepAgent to use additional paths from the primary database to support Multi-Path Replication™.

See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication*.

Configuring RepAgent Using Command Line Options

There are several basic steps for configuring RepAgent from the command line.

Prerequisites

Task

1. Define the local Adaptive Server using **sp_addserver**.
2. Enable the RepAgent feature on Adaptive Server using **sp_configure**.
3. Enable the RepAgent feature for each database using **sp_config_rep_agent**.
4. Enable log transfer on Replication Server using **alter connection**.
5. Start the RepAgent on Adaptive Server using **sp_start_rep_agent**.

Define the Local Adaptive Server

If you are starting Adaptive Server for the first time, you must execute the Adaptive Server system procedure **sp_addserver** to add an entry for the local server to the Adaptive Server `sysservers` table.

See the *Adaptive Server Enterprise Reference Manual* for information about using **sp_addserver**.

Enabling RepAgent on Adaptive Server

Use **sp_configure** to enable RepAgent on Adaptive Server.

You need to perform this task only once at each Adaptive Server.

sp_configure 'enable rep agent threads' is a dynamic option. It takes effect immediately. However, you may want to restart Adaptive Server after enabling RepAgent so that Adaptive Server allocates a fixed number of dedicated static process structures for the thread. Otherwise, RepAgent borrows process structures from the pool dedicated to user connections.

Log in to Adaptive Server and at the **isql** prompt, enter:

```
sp_configure 'enable rep agent threads', 1
```

Enabling RepAgent at the Primary Database

Execute **sp_config_rep_agent** to enable the RepAgent for each primary database and set default values for RepAgent configuration parameters.

You can reset the default values at a later time. In this example, *dbname* is the name of the database for which you are enabling RepAgent, *repserver_name* is the Replication Server to

which RepAgent connects, and *repserver_username* and *repserver_password* are the name and password RepAgent uses to log in to Replication Server.

Note: Make sure that *repserver_username* is a valid Replication Server user and that it has Replication Server **connect source** permission. Try out the user name and password at the Replication Server before you use **sp_config_rep_agent**.

Log in to Adaptive Server. At the **isql** prompt, and enter:

```
use dbname
go
sp_config_rep_agent dbname, enable, 'repserver_name',
    'repserver_username', 'repserver_password'
```

See also

- *Configuring RepAgent* on page 113

Enabling Log Transfer for the Primary Database

Use **set log transfer on** to enable log transfer for each connection between Replication Server and a primary database.

Note: You must create a database connection between Replication Server and the data server using **rs_init** or **create connection** before you can turn on log transfer. See the *Replication Server Configuration Guide* for your platform for information about creating connections using **rs_init**.

Turn on log transfer for the database connection to the primary database using **alter connection**. For example, at the Replication Server, enter:

```
alter connection to TOKYO_DS.pubs2
    set log transfer on
```

See also

- *Manage Database Connections* on page 189

Configuring RepAgent

Use **sp_config_rep_agent** to change the default RepAgent configuration parameters.

The default RepAgent configuration parameters are set after enabling RepAgent using **rs_init** or **sp_config_rep_agent**. Configuration parameters affecting RepAgent are stored in the *sysattributes* table of the database for which RepAgent is enabled. You must restart RepAgent for the new parameters to take effect. There are also network security configuration parameters for RepAgent if your system supports network-based security.

To configure RepAgent:

1. Log in to Adaptive Server and specify the database.
For example:

Manage RepAgent and Support Adaptive Server

```
use dbname  
go
```

2. Execute **sp_config_rep_agent** once for each parameter you want to configure. For example, to change the maximum number of log records sent to Replication Server in a batch to 2000:

```
sp_config_rep_agent dbname, 'scan batch size', '2000'  
go
```

3. Restart RepAgent for the new parameter to take effect.

```
sp_start_rep_agent dbname  
go
```

In the *Replication Server Reference Manual* see:

- *Adaptive Server Commands and System Procedures*
- *Configuration Parameters Affecting RepAgent*

.

See also

- *Manage Network-based Security* on page 261
- *Starting RepAgent* on page 115

Master Key and rs password

Set the master key password and **rs password** attributes to continue replication.

In Adaptive Server, when you create the `syb_extpasswdkey` service key with the master key and you have not set the master key password in memory manually or automatically, the Adaptive Server RepAgent is blocked at startup and **sp_who** shows "MASTER KEY SLEEP" until you set the master key password. Each replication path has one **rs password** attribute that RepAgent uses to log in to Replication Server. When you drop the `syb_extpasswdkey` service key, Adaptive Server resets all the existing RepAgent **rs password** attributes. If you enter **sp_encryption helpextpasswd**, you see "Needs Reset". You must reset all the **rs password** attributes to continue replication.

See *Adaptive Server Enterprise > Encrypted Columns Users Guide > Securing External Passwords and Hidden Text > Service Keys*.

Starting RepAgent

Use **sp_start_rep_agent** to start RepAgent on Adaptive Server.

Prerequisites

There must be an entry for Replication Server in the interfaces file of the Adaptive Server.

Task

Normally, you need to start a RepAgent thread only if you:

- Reconfigure the RepAgent parameters.
- Explicitly shut down the RepAgent.

RepAgent starts automatically when Adaptive Server restarts if you had:

- Started RepAgent at least once with **sp_start_rep_agent** and not stopped RepAgent with **sp_stop_rep_agent**.
- Set **auto start** to true with **sp_config_rep_agent**.

Note: RepAgent can be restarted only if its associated database is fully recovered and online and log transfer is on for the connection to the primary database.

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures* for detailed information about each option of **sp_start_rep_agent** and **sp_config_rep_agent**.

1. Log in to Adaptive Server
2. At the **isql** prompt, enter **sp_start_rep_agent**.
For example, to enable RepAgent for the pubs2 database:

```
sp_start_rep_agent pubs2
```

(Optional) To automatically start RepAgent whenever Adaptive Server restarts:

```
sp_config_rep_agent 'auto start', 'true'
```

Stopping RepAgent

To shut down RepAgent, log in to Adaptive Server and execute **sp_stop_rep_agent**.

When RepAgent restarts, it scans records starting with the oldest transaction, but it only sends records following the last one processed. Replication Server determines if there are any duplicate records and discards the duplicates.

Once RepAgent has been shut down with **sp_stop_rep_agent**, it does not automatically start up when the database comes online during data server startup unless you previously set **auto**

start to true with **sp_config_rep_agent**. Otherwise, you must execute **sp_start_rep_agent** to start RepAgent and resume automatic start-up.

For example, to stop RepAgent, enter:

```
sp_stop_rep_agent pubs2
```

If you shut down RepAgent in this way, Adaptive Server shuts down RepAgent gracefully at the end of the current batch of transactions.

You can also shut down RepAgent immediately using the **nowait** option. For example:

```
sp_stop_rep_agent pubs2, nowait
```

If you shut down RepAgent with the **nowait** option, Adaptive Server terminates the RepAgent without waiting for currently executing operations to finish.

Disable RepAgent

Use **sp_config_rep_agent** to disable RepAgent.

Note: You should disable RepAgent only when you change the replicate database to a primary database, or downgrade Replication Server to an earlier version.

Before disabling RepAgent using **sp_config_rep_agent**, you must first shut it down using **sp_stop_rep_agent**.

Normally, when you disable RepAgent, the process also disables the secondary truncation point. For example:

```
sp_config_rep_agent pubs2, 'disable'
```

Once the secondary truncation point is disabled, the log can get truncated past the secondary truncation point.

To disable RepAgent but keep the secondary truncation point, use the **preserve secondary truncpt** option.

```
sp_config_rep_agent pubs2, 'disable', 'preserve  
secondary truncpt'
```

Disable RepAgent in this way to disable RepAgent momentarily.

If you are changing the primary to a replicate database, you must also turn log transfer off. After disabling RepAgent, turn log transfer off using **alter connection**.

For example, log in to Replication Server and enter:

```
alter connection to TOKYO_DS.pubs2  
set log transfer off
```

Configure RepAgent for Network Security

You can secure the pathway between RepAgent and Replication Server using network-based security features.

Using `sp_config_rep_agent`, you can change settings for:

- The active security mechanism
- Unified login
- Mutual authentication
- Message confidentiality
- Message integrity
- Message replay detection
- Message origin check
- Message out of sequence check

See also

- *Manage Network-based Security* on page 261

Manage Log Transfer Activity

Use the log transfer commands to suspend and resume log transfer if you perform recovery, troubleshooting, or diagnostic tasks.

The log transfer commands are:

- **resume log transfer** and **suspend log transfer**
- **alter connection ... set log transfer on/off**

Note: RepAgent cannot connect to Replication Server unless log transfer has first been set on using **alter connection**.

See *Replication Server Administration Guide Volume 2 > Replication System Recovery* for information about starting the RepAgent thread in recovery mode so that it can replay database and transaction dumps.

Suspend Log Transfer

Execute **suspend log transfer** to disconnect one or all RepAgents and prevent RepAgents from connecting to Replication Server.

Log transfer to Replication Server remains suspended until you resume it using the **resume log transfer** command.

suspend log transfer command records information in the RSSD, so if you shut down Replication Server and restart it, log transfer to that Replication Server remains suspended.

Note: Suspending log transfer is the first step in quiescing the replication system.

To suspend log transfer, log in to Replication Server and execute **suspend log transfer** at the **isql** prompt by entering:

```
suspend log transfer from {data_server.database | all}
```

where:

- *data_server* – the data server with the database for which log transfer is to be suspended.
- *database* – the database for which log transfer is to be suspended.
- **all** – instructs Replication Server to suspend log transfer from all RepAgent and disallow future connections for all RepAgent.

These examples demonstrate the use of **suspend log transfer** .

- Suspend log transfer for the database named `pubs2`, managed by the `TOKYO_DS` data server:

```
suspend log transfer from TOKYO_DS.pubs2
```

- Suspend log transfer to the current Replication Server from all RepAgents:

```
suspend log transfer from all
```

In both examples, after the command is executed, affected RepAgent are not shut down and may continue to send some messages to Replication Server. To shut down a RepAgent immediately, log in to Adaptive Server and enter **sp_stop_rep_agent**, with the name of the database for which RepAgent is enabled, and the **nowait** option.

See also

- *Quiesce Replication Server* on page 105

Resume Log Transfer

To reconnect RepAgent to a Replication Server, log in to the Replication Server and enter the **resume log transfer** command at the **isql** prompt.

```
resume log transfer from {data_server.database | all}
```

where:

- *data_server* – the data server with the database for which log transfer is to be resumed.
- *database* – the database for which log transfer is to be resumed and for which the RepAgent connection is to be allowed.
- **all** – allows all RepAgents to connect to this Replication Server.

These examples demonstrate the use of **resume log transfer**:

- Resume log transfer for the database named pubs2, managed by the TOKYO_DS data server:

```
resume log transfer from TOKYO_DS.pubs2
```

- Resume log transfer to this Replication Server from all RepAgents:

```
resume log transfer from all
```

Use alter connection and the set log transfer Option

Use **alter connection** with the **set log transfer** option to shut down log transfer.

To shut down log transfer, turn the **set log transfer** option off. For example:

```
alter connection to TOKYO_DS.pubs2
  set log transfer off
```

When log transfer is off, Replication Server removes the DIST thread, and RepAgent can no longer log in to Replication Server.

When Replication Server no longer recognizes the primary database, you must reestablish this connection using **rs_init** or **create connection** before you can use **alter connection** to set log transfer on.

To set log transfer on, turn the **set log transfer** option on. For example:

```
alter connection to TOKYO_DS.pubs2
  set log transfer on
```

Review RepAgent Status and Configuration Information

You can monitor RepAgent and see the values of configuration parameters using commands and system procedures. You can also use the Adaptive Server plug-in to Sybase Central to monitor RepAgent.

View RepAgent Information

Use **sp_help_rep_agent** at Adaptive Server to monitor the RepAgent.

sp_help_rep_agent displays information about:

- Recovery – status and other information when you are restoring a database.
- Configuration parameters – the current settings for RepAgent configuration parameters for both single and multiple replication paths.
- Process – information about the RepAgent process, including state, sleep status, number of unsuccessful connection retries (if any), and the number of the last error message, for both single and multiple replication paths.
- Send buffers - the number of send buffers that you have allocated to RepAgent.
- Scanned transactions – information about the current batch of log transactions: start, end, and current markers; the number of records in the batch; and the oldest transaction.

Manage RepAgent and Support Adaptive Server

- Security – the current settings of the network-based security mechanism.
- All – all of the above information.

Log in to Adaptive Server and execute **sp_help_rep_agent** at the **isql** prompt:

```
sp_help_rep_agent [dbname[, 'recovery' | 'config' | 'process' |  
'send' | 'scan' | 'security' | 'all']]
```

dbname is the name of the database for which the RepAgent is enabled.

You can view current status information for one or all options, for example:

- To display information about the RepAgent process, log in to Adaptive Server and enter:

```
sp_help_rep_agent pubs2, 'process'
```

- To display information about the RepAgent log scanning, enter:

```
sp_help_rep_agent pubs2, 'scan'
```

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures* for detailed syntax, usage information, and example output for **sp_help_rep_agent**.

View RepAgent Configuration Parameter Values

To view a list of default, current, and runtime configuration parameter values for a particular RepAgent, log in to Adaptive Server and execute **sp_config_rep_agent** without options.

For example:

```
sp_config_rep_agent pubs2
```

If you do not specify a database name, **sp_config_rep_agent** displays configuration values for all RepAgent-enabled databases.

To view values for a specific parameter, include the parameter name. For example:

```
sp_config_rep_agent pubs2, 'scan batch size'
```

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_config_rep_agent*.

View RepAgent Thread Information

Execute **sp_who** to view the RepAgent thread status on Adaptive Server.

In the display output, Adaptive Server shows the RepAgent information in rows with “REP AGENT” in the “cmd” column.

For example, **sp_who** might display this row for RepAgent:

```
fid spid status loginame origname hostname blk_spid dbname cmd  
block_xloid  
-----  
...  
0 23 background NULL NULL 0 pubs2 REP AGENT 0  
...
```

See the *Adaptive Server Enterprise Reference Manual* for detailed syntax and usage information for **sp_who**.

To view RepAgent thread user status on Replication Server, execute **admin who**. Replication Server displays RepAgent thread user information in rows with “REP AGENT” in the “name” column.

See *Replication Server Reference Manual > Replication Server Commands* for more information about **admin who** and output examples.

Check Log Files for RepAgent Information and Error Messages

Obtain RepAgent error and information messages that are recorded in the Adaptive Server errorlog file.

See the *Adaptive Server Enterprise System Administration Guide* for more information about the Adaptive Server error log.

For example, starting RepAgent generates this message in the Adaptive Server errorlog:

```
00:00000:00022:2003/09/18 12:16:39.15 server Started  
RepAgent on database, 'pubs2' (dbid = 4).
```

Stopping RepAgent generates this message:

```
00:00000:00022:2003/09/18 12:17:17.07 server Shutting  
down RepAgent for database, 'pubs2' (dbid=4).
```

Use Counters to Monitor RepAgent Performance

Adaptive Server provides several counters for monitoring RepAgent performance. You can monitor RepAgent performance data using **sp_sysmon**.

Invoking **sp_sysmon** clears all accumulated data from the set of counters to be used during the sample interval. At the end of the sample interval, the procedure reads the values in the counters, prints a report, and stops executing.

You can direct **sp_sysmon** to print information for RepAgent counters only or for all Adaptive Server counters. **sp_sysmon** displays RepAgent counter information for each database.

See the *Adaptive Server Enterprise Performance and Tuning Guide* for complete usage and syntax information for **sp_sysmon**.

See *Replication Server Administration Guide Volume 2 > Use Counters to Monitor Performance* to use counters to monitor Replication Server activity.

Invoke sp_sysmon

You can invoke **sp_sysmon** by using fixed time intervals, or by using the **begin_sample** and **end_sample** parameters.

Fixed Time Intervals

Use fixed time intervals to provide a sample for a specified number of minutes.

For example, you can run **sp_sysmon** for 10 minutes and print information for all counters:

```
sp_sysmon "00:10:00"
```

To print only the RepAgent section of the report, enter:

```
sp_sysmon "00:10:00", repagent
```

begin_sample and end_sample

Use **begin_sample** and **end_sample** to invoke **sp_sysmon** to start and end sampling, issue queries, and print results at any point in time.

For example, to start and end the sample for the RepAgent group of counters, enter:

```
sp_sysmon begin_sample
go
execute procl
go
sp_sysmon end_sample, repagent
```

Sample Output from sp_sysmon of RepAgent Activity

View a sample output of RepAgent counter activity from **sp_sysmon**.

Sample output from sp_sysmon

```
Replication Agent
-----
Replication Agent: pubs2
Replication Server: NY_RS
```

	per sec -----	per xact -----	count -----	% of total -----
Log Scan Summary				
Number of Log Scans	n/a	n/a	1	n/a
Amount of Time				
for Log Scan (ms)	n/a	n/a	3822	n/a
Longest Time				
for Log Scan (ms)	n/a	n/a	3822	n/a
Average Time				
per Log Scan (ms)	n/a	n/a	3822	n/a
Log Scan Activity				
Updates	n/a	n/a	5	n/a

Manage RepAgent and Support Adaptive Server

Inserts	n/a	n/a	5	n/a
Deletes	n/a	n/a	5	n/a
Store Procedures	n/a	n/a	0	n/a
DDL Log Records	n/a	n/a	0	n/a
Writetext Log Records	n/a	n/a	0	n/a
Text/Image Log Records	n/a	n/a	10	n/a
CLRs	n/a	n/a	0	n/a
Checkpoints Processed	n/a	n/a	0	n/a
Transaction Activity				
Opened	n/a	n/a	7	n/a
Committed	n/a	n/a	7	n/a
Aborted	n/a	n/a	0	n/a
Delayed Commit	n/a	n/a	0	n/a
Maintenance User	n/a	n/a	0	n/a
Log Extension Wait				
Count	n/a	n/a	3	n/a
Amount of time (ms)	n/a	n/a	7822	n/a
Longest Wait (ms)	n/a	n/a	5110	n/a
Average Time (ms)	n/a	n/a	2607.3	n/a
Schema Cache				
Usage				
Max Ever Used	n/a	n/a	0	n/a
Schemas reused	n/a	n/a	0	n/a
Forward Schema Lookups				
Count	n/a	n/a	0	n/a
Total Wait (ms)	n/a	n/a	0	n/a
Longest Wait (ms)	n/a	n/a	0	n/a
Average Time (ms)	n/a	n/a	0.0	n/a
Backward Schema Lookups				
Count	n/a	n/a	0	n/a
Total Wait (ms)	n/a	n/a	0	n/a
Longest Wait (ms)	n/a	n/a	0	n/a
Average Time (ms)	n/a	n/a	0.0	n/a
Truncation Point Movement				
Moved	n/a	n/a	0	n/a
Gotten from RS	n/a	n/a	0	n/a
Connections to Replication Server				
Success	n/a	n/a	0	n/a
Failed	n/a	n/a	0	n/a
Network Packet Information				
Packets Sent	n/a	n/a	6	n/a
Full Packets Sent	n/a	n/a	2	n/a
Largest Packet	n/a	n/a	2048	n/a
Amount of Bytes Sent	n/a	n/a	7695	n/a
Average Packet	n/a	n/a	1282.5	n/a
I/O Wait from RS				
Count	n/a	n/a	6	n/a

Amount of Time (ms)	n/a	n/a	766	n/a
Longest Wait (ms)	n/a	n/a	206	n/a
Average Wait (ms)	n/a	n/a	127.7	n/a

Description of Sample Output of RepAgent Counter Activity

Each section in the sample output of RepAgent counter activity from **sp_sysmon** describes a different activity.

Log Scan Summary

RepAgent scans all records in the transaction log, but not all scanned records need to be processed and sent to Replication Server. For example, RepAgent does not send records generated by data manipulation language (DML) on tables not marked for replication.

Log Scan Summary reports the number of log records:

- RepAgent has scanned
- RepAgent has processed and sent to Replication Server

Log Scan Activity

Log Scan Activity provides information about the different kinds of log records processed by RepAgent and sent to the Replication Server.

Log Scan Activity reports the number of:

- Rows affected by **update** statements
- Rows affected by **insert** statements
- Rows affected by **delete** statements
- Stored procedure executions
- Log records containing DDL to be replicated
- Log records processed generated by a **WriteText** command
- DML log records processed for a table with `text`, `unitext`, or image data
- Compensation log records (CLRs), which are generated when a transaction is partially or fully rolled back
- Checkpoint log records indicate that there was an active transaction at the time this log record was written.

Transaction Activity

Transaction Activity reports the number of transactions :

- Opened in the primary database
- Committed
- Aborted

- Found in prepare state
- Opened by the maintenance user

Log Extension Wait

During normal processing, RepAgent reaches the end of the transaction log. It then waits until further activity resumes in the primary database.

Log Extension Wait reports:

- The number of times RepAgent waited for extensions to the transaction log
- The total amount of time, in milliseconds (ms), that RepAgent waited for log extensions
- The longest amount of time, in ms, that RepAgent waited for log extensions
- The average amount of time, in ms, that RepAgent waited for log extensions

Schema Cache

When the structure of an object marked for replication is modified—by **alter table**, for example—Adaptive Server must log special records in the transaction log that later on will help RepAgent identify the correct schema for the object.

Schema Cache reports schema activity and RepAgent activity scanning forward and backward in the transaction log looking for object schema changes.

- Usage

Usage reports:

- The maximum number of active schemas in the Schema Cache since the last restart of Replication Agent
- The number of times that a schema had to be removed from the Schema Cache to free space for a new one

- Forward Schema Lookups

Forward Schema Lookups reports:

- The number of times RepAgent performed forward scans
- The total amount of time, in ms, that RepAgent spent performing forward scans
- The longest amount of time, in ms, that RepAgent spent performing a forward scan
- The average amount of time, in ms, that RepAgent spent performing a forward scan

- Backward Schema Lookups

RepAgent performs a backward scan when DDL is performed inside a transaction.

Backward Schema Lookups reports:

- The number of times RepAgent spent performing backward scans
- The total amount of time, in ms, that RepAgent performed backward scans
- The longest amount of time, in ms, that RepAgent spent performing a backward scan
- The average amount of time, in ms, that RepAgent spent performing a backward scan

Truncation Point Movement

Truncation Point Movement reports:

Manage RepAgent and Support Adaptive Server

- The number of times RepAgent moved the secondary truncation point
- The number of times RepAgent asked Replication Server for a new truncation point

Connections to Replication Server

Connections to Replication Server reports:

- The number of successful connections to Replication Server
- The number of unsuccessful connections to Replication Server

Network Packet Information

Network Packet Information reports:

- The number of packets sent to Replication Server
- The number of full packets sent to Replication Server
- The largest packet sent to Replication Server
- The number of bytes sent to Replication Server
- The average packet size

I/O Wait from Replication Server

After RepAgent generates LTL, RepAgent sends it to Replication Server. To do this, it uses Open Client capabilities.

I/O Wait from Replication Server reports:

- The number of times RepAgent has sent a batch to Replication Server
- The total amount of time, in ms, that RepAgent has spent processing results from Replication Server
- The longest elapsed time, in ms, that RepAgent has spent processing results from Replication Server
- The average elapsed time, in ms, that RepAgent has spent processing results from Replication Server

Support for Extended Limits

Replication Server version 12.5 and later supports extended limits for replication definitions.

Supported extended limits are:

- More columns, to a maximum of 1024
- Wide columns and parameters, to a maximum of 32768 bytes
- Wide data rows to the width of the data page on the data server
- Wide messages larger than 16K

If the Replication Server site version is 12.5 or later, Replication Server sets the LTL version automatically to 400 or higher. If RepAgent is running on Adaptive Server 12.5 or later,

RepAgent sends data with extended limits only if Replication Server specifies an LTL version of 400 or higher at connect source time.

If the Replication Server site version is 12.1 or earlier, the LTL version is earlier than 400. If RepAgent is running on Adaptive Server 12.5 or later, Sybase recommends that you do not send extended-limits data to Replication Server 12.1 and earlier. You can specify how RepAgent handles extended-limits data by using the **data limits filter mode** parameter with **sp_config_rep_agent**

See also

- *Configuring RepAgent* on page 113

Support for Longer Identifiers

Replication Server version 15.0 and later increases the maximum length of replication object identifiers to 255 bytes.

Affected replication object identifiers include:

- Table name and column name
- Stored procedure name and parameter name
- Functions and parameters – for function replication definitions and internal use only
- Function string name
- Replication definitions – including table replication definitions, function replication definitions, and database replication definitions
- Article name
- Publication name

If the Replication Server site version is 15.0, Replication Server sets the LTL version automatically to 700. If RepAgent is running on Adaptive Server 15.0 or later, RepAgent sends data with extended size only if Replication Server specifies an LTL version of 700 or higher at connect source time.

If the Replication Server site version is 12.6 or earlier, the LTL version is earlier than 700. If RepAgent is running on Adaptive Server 15.0 or later, Sybase recommends that you do not send data with longer identifiers to Replication Server 12.6 and earlier.

You can specify how RepAgent handles data with longer identifier by using the **data limit filter mode** parameter with **config_rep_agent**.

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.

See also

- *Configuring RepAgent* on page 113

Support for bigdatetime and bigtime Datatypes

Replication Server supports replication of the `bigdatetime` and `bigtime` datatypes included with Adaptive Server.

You can replicate these datatypes to replicate databases and warm standby databases by specifying the datatypes in replication definitions, function replication definitions, and subscriptions.

`bigdatetime` and `bigtime` allow Adaptive Server to store data and time data up to microsecond precision. `bigdatetime` corresponds to the `TIMESTAMP` datatype, and `bigtime` corresponds to the `TIME` datatype in Sybase IQ.

See *Replication Server Reference Manual > Topics > Datatypes > Date/time, and date and time Datatypes* for descriptions of the datatypes.

You can use `rs_helprep` to display information about `bigdatetime` and `bigtime`.

`rs_subcmp` supports `bigdatetime` and `bigtime`.

See the *Replication Server Reference Manual* for descriptions of the commands.

These examples show how to use `bigdatetime` and `bigtime` in a replication definition, a function replication definition, and a subscription.

In these examples:

- PDS – primary data server
- pdb1 – primary database
- RDS – replicate data server
- rdb1 – replicate database
- tb1 – table
- col1, col2, col3 – columns
- **rep1** – replication definition
- **func1** – function replication definition
- **sub1** – subscription

Replication Definition

```
create replication definition rep1
with primary at PDS.pdb1
with all tables named tb1
(col1 int, col2 bigdatetime, col3 bigtime)
primary key (col1)
```

Function Replication Definition

```
create function replication definition func1
with primary at PDS.pdb1
```

```
(@par1 int, @par2 bigdatetime, @par3 bigtime)
searchable parameters (@par1)
```

Subscription

```
create subscription sub1 for repl
with replicate at RDS.rdb1
where col3 = '14:20:00.010101'
without materialization
```

System Table Support for bigdatetime and bigtime

The `rs_columns`, `rs_datatypes`, and `rs_objects` system tables are extended to support the replication of the `bigdatetime` and `bigtime` datatypes.

rs_columns

Column	Data-type	Description
coltype	tinyint	Datatype of the column or parameter: <ul style="list-style-type: none"> • 35 – bigdatetime • 36 – bigtime

rs_datatype

Column	Data-type	Description
base_coltype	tinyint	ID of base datatype for the datatype. Can be: <ul style="list-style-type: none"> • 35 – bigdatetime • 36 – bigtime

rs_objects

Column	Data-type	Description
attributes	int	Mask, can be one or more of the following: <ul style="list-style-type: none"> • 0x02 – replication definition has bigdatetime or bigtime columns and can be propagated only to Replication Server 15.5 or later.

Mixed-Version Information for bigdatetime and bigtime

Only Adaptive Server versions 15.5 and later support `bigdatetime` and `bigtime`.

If the primary data server is at least Adaptive Server 15.5, and:

- Primary and replicate Replication Server are version 15.5 or later, and the replicate Adaptive Server does not support the datatypes, you can create a replication definition containing a mapping for each of the two datatypes to the `varchar` datatype. Alternatively, use the `varchar` datatype instead of the two datatypes in the replication definition.
- Primary Replication Server is version 15.5 or later, and the replicate Replication Server and Adaptive Server do not support the datatypes, use the `varchar` datatype instead of the two datatypes in the replication definition.
- Primary and replicate Replication Server, and the replicate Adaptive Server do not support the datatypes, RepAgent automatically sends the `varchar` datatype to Replication Server.

Adaptive Server Shared-disk Cluster Support

Replication Server and RepAgent thread both support the Adaptive Server shared-disk cluster environment.

In a Sybase shared-disk cluster, a database can be either a replication source or a replication destination. You can perform all of the tasks, such as configuring RepAgent or marking tables for replication, from any instance in the cluster. Replication status is coherent across the entire cluster.

When adding new connections from or to an Adaptive Server cluster environment, the *servername* in the connection syntax must be the *clustername* and not the *instancename*. Use `select @@servername` to retrieve the *clustername*.

By default, the RepAgent starts on the cluster coordinator; However, you can configure it to start on any instance in the cluster. For example, to configure the RepAgent on the primary database `pdb` to always start on the “ase2” instance, enter:

```
sp_config_rep_agent pdb, "cluster instance name", "ase2"
```

For a new configuration to take effect, restart the RepAgent using `sp_start_rep_agent`. To return to the default behavior with the RepAgent starting on the cluster coordinator, enter:

```
sp_config_rep_agent pdb, "cluster instance name",  
"coordinator"
```

When an instance starts, it checks if there are RepAgents configured to start on its node. If there are, and if the database is marked to start automatically, the RepAgent starts.

When the cluster coordinator starts, it also starts all RepAgents that are not configured to start on a specific instance. If the coordinator node fails, or is stopped with a graceful shutdown, a RepAgent starts on the new coordinator node.

If the RepAgent is configured to start on an instance other than the coordinator node, and this instance fails, the RepAgent starts on the coordinator.

See the *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_config_rep_agent* for information about the **cluster instance name** configuration parameter.

Deferred Name Resolution

Deferred name resolution lets you create stored procedures in Adaptive Server without resolving the objects used internally by these stored procedures. Adaptive Server postpones the object resolution phase to the first time you execute the stored procedure in Adaptive Server.

Stored procedures execute normally when you execute them after the first time.

In Replication Server versions earlier than 15.5, you can set up a warm standby application and enable **sp_reptostandby** at the active database to allow replication of supported data definition language (DDL) commands to the standby database. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications*.

However on a standby database or a replicate database in a non-Warm Standby environment, you cannot create a stored procedure that references a temporary table because Replication Server does not replicate temporary tables. The create stored procedure process must resolve the objects used internally by the stored procedure. However since there is no temporary table in the replicate or standby database, Replicate Server does not create the stored procedure in the replicate or standby database.

With support for deferred name resolution, Replication Server allows the replication of stored procedures that refer to temporary tables, tables that do not exist, and procedures that do not exist, to replicate or standby databases.

See *Adaptive Server Enterprise Transact-SQL Guide*, *Adaptive Server Enterprise System Administration Guide: Volume 1*, and *Adaptive Server Enterprise Reference Manual: Commands*.

Configure Replication Server to Support Deferred Name Resolution

To enable or disable deferred name resolution in Replication Server, use the **deferred_name_resolution** parameter with **alter connection** for a physical connection or **alter logical connection** for a logical connection in a warm standby application.

For example, to enable deferred name resolution on the *pubs2* warm standby database of the SYDNEY_DS data server:

```
alter logical connection to SYDNEY_DS.pubs2  
set deferred_name_resolution to 'on'
```

After you execute **deferred_name_resolution** with **alter connection** or **alter logical connection**, suspend and resume the connection. By default, **deferred_name_resolution** is off.

Note: Enable deferred name resolution support in Adaptive Server before you configure deferred name resolution support in Replication Server.

Adaptive Server Data Compression

Replication Server supports the Adaptive Server data compression feature.

Adaptive Server data compression lets you use less storage space for the same amount of data, reduce cache memory consumption, and improve performance by lowering I/O demands. Adaptive Server can compress large object (LOB) datatypes such as `text`, `image`, and `unibase`, and non-LOB datatypes. See the *Adaptive Server Enterprise Compression Users Guide*.

Adaptive Server stores data in-row or off-row. Adaptive Server stores in-row, data columns that are located contiguous to all other columns for that row. Adaptive Server stores LOB data off-row in other locations because of the size of the data. There is an in-row pointer to the actual location of off-row data.

System Requirements

- Adaptive Server – version 15.7 ESD #1 and later for both primary and replicate databases.
- Replication Server – version 15.7.1 and later for the primary and replicate Replication Server.

Replication Server Support for Compressed Data

Replication Server decompresses compressed Adaptive Server data before sending it to the replicate database.

Support for replication of compressed data between an Adaptive Server primary database and an Adaptive Server or heterogeneous replicate database depends on whether the data is in-row or off-row:

- Replication Server supports replicating in-row non-LOB compressed data between databases.
- Replication Server supports replicating compressed off-row LOB columns between an Adaptive Server primary database and an Adaptive Server or heterogeneous replicate database. Replication Server decompresses LOB data before sending it to a heterogeneous replicate database. Replication Server sends compressed LOB data directly to an Adaptive Server replicate database only if the replicate has the same LOB schema, character set,

endianness, version, and page size as the primary Adaptive Server database. Otherwise, Replication Server decompresses LOB data before sending it to the replicate Adaptive Server database.

The Replication Server **dsi_do_decompression** parameter must be on for Replication Server to decompress LOB data. This parameter is set to on in connection profiles for connecting to a heterogeneous replicate database.

For the LOB data decompression feature, the primary and replicate Replication Servers and the primary Adaptive Server must be version 15.7.1 SP100 or later.

Subscription Materialization of LOB Compressed Data

Adaptive Server 15.7 and later supports LOB compression.

Support for subscription materialization of LOB compressed data depends on:

- How you specify the column datatype in the replication definition
- Whether there is at least one version 15.7.1 SP100 or later Replication Server, and the position of this Replication Server relative to the other Replication Server in a replication system with multiple Replication Servers

If the Replication Server version is earlier than 15.7.1 SP100, Replication Server supports materialization of LOB compressed and uncompressed data if the datatypes of the LOB columns you define in the replication definition are the same as the datatypes of the corresponding LOB columns in the primary table schema.

Table 8. Replication Server Versions Earlier Than 15.7.1 SP100

Column Datatype in Primary Table Schema	Column Datatype in Replication Definition	Materialization Supported
image	image	Yes
text	text	Yes
unitext	unitext	Yes
image	text or unitext	No
text	image or unitext	No
unitext	text or image	No

If the Replication Server version is 15.7.1 SP100 and later, Replication Server supports materialization of LOB:

- Compressed data – if you define the datatypes of the LOB columns in the replication definition as image

Column Datatype in Primary Table Schema	Column Datatype in Replication Definition	Materialization Supported
image	image	Yes
text	image	Yes
unitext	image	Yes

- Uncompressed data – if the datatypes of the LOB columns you define in the replication definition are the same as the datatypes of the corresponding LOB columns in the primary table schema.

Column Datatype in Primary Table Schema	Column Datatype in Replication Definition	Materialization Supported
image	image	Yes
text	text	Yes
unitext	unitext	Yes
text	image	Yes Warning! Although materialization completes, do not define the column datatype as <code>image</code> in the replication definition as this results in data corruption during replication. See <i>Replication Server Troubleshooting Guide > RepAgent Problems > Errors from the Replication Server > Error 32044 > Solution</i> to fix data corruption if you have incorrectly defined the datatypes in the replication definition.

Column Datatype in Primary Table Schema	Column Datatype in Replication Definition	Materialization Supported
unitext	image	Yes Warning! Although materialization completes, do not define the column datatype as image in the replication definition as this results in data corruption during replication. See <i>Replication Server Troubleshooting Guide > RepAgent Problems > Errors from the Replication Server > Error 32044 > Solution</i> to fix data corruption if you have incorrectly defined the datatypes in the replication definition.

Example of a replication definition specifying the au_photo LOB column as image datatype:

```
create replication definition pubs_copy_rep1
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, au_photo image)
primary key (pub_id)
```

Example of a replication definition specifying the c_image, c_text, and c_unitext columns as image, text, and unitext datatypes respectively, to match the original datatypes in the pubs2 primary database:

```
create replication definition pubs_copy_rep2
with primary at TOKYO_DS.pubs2
with all tables named lob (c_key integer, c_image image null, c_text
text null, c_unitext unitext null )
primary key (c_key)
```

If you incorrectly define the column datatypes for the LOB columns in the replication definition, subscription materialization does not proceed and RepAgent shuts down. See *Replication Server Troubleshooting Guide > RepAgent Problems > Errors from the Replication Server > Error 32044* to purge the corrupted data that was replicated and to resume replication.

Maintaining Data Integrity

To maintain data integrity, use a customized function string to remove undesired data out of the queues if you do not disable replication of off-row compressed LOB columns.

1. Disable replication of compressed LOB columns:

```
sp_setrepcol table_name, lob_column_name, 'do_not_replicate'
```

2. If the DSI thread shuts down, use a customized function string to remove undesired data from the queue.

For example, using the table schema in the `mytab5` table and a replication definition named **reptab5**, create a function string named **reptab5.rs_insert** based on the **rs_insert** function:

- Table schema:

```
create table mytab5
(c1 int primary key, c2 text null compressed = 5)
```

- Replication definition:

```
create replication definition reptab5
with primary at repl3_18540.pdb
with all tables named mytab5
(c1 int, c2 image null)
primary key (c1)
```

- Customized function string:

```
alter function string reptab5.rs_insert
for rs_sqlserver_function_class
output language
'insert mytab5(c1, c2) values(?c1!new? , '''' )'
```

Support for Incremental Data Transfer

With Adaptive Server 15.5, you can transfer data incrementally from a table instead of having to transfer the whole table from one Adaptive Server to another. Replication Server supports the data definition language related to the Adaptive Server incremental data transfer feature.

Replication proceeds normally for data modification operations performed on a replicate table marked for incremental transfer.

If you load a replicate table with the **transfer table** command, and the table has an unique index command and the data on the incremental transfer already exists on the table, Adaptive Server internally converts an **insert** command into an **update** command.

The **transfer table** command applies only to the data server and database where you initiated the transfer the first time.

If you mark tables for incremental transfer in the active database within a warm standby or MSA environment, and then switch to the standby database if the active database terminates, incremental data transfer may not resume correctly at the standby database. This is because,

unlike the active database, the standby database does not have a record of the incremental data transfer activity. Therefore, you must initialize the incremental data transfer on the standby database too.

See *Adaptive Server Enterprise Transact-SQL Users Guide > Adding, Changing, Transferring, and Deleting Data > Transferring data Incrementally*.

Predicated Privileges

There are several requirements for Replication Server support for Adaptive Server predicated privileges.

Adaptive Server 15.7 ESD #2 includes predicated privileges. Replication Servers in a mixed-version environment support predicated privileges.

To use predicated privileges in a replication system, the maintenance user on the replicate Replication Server must have nonpredicated privileges on the replicate tables and databases.

See *Predicated Privileges* in the *Adaptive Server Enterprise 15.7 ESD #2 New Features Summary*.

Stored Procedure Replication with Predicated Privileges

Executing a replicated stored procedure on the primary database by a user with predicated privileges for the DML statements in the procedure, may cause inconsistency between the primary and replicate data.

Suppose that you, as the Adaptive Server and Replication Server administrator, grant the **update** privilege on the `accounts` table to Marc:

```
grant update on accounts to Marc (1)
where owner = 'Marc'
```

In addition, create the **balance_limit** stored procedure on the primary database and mark **balance_limit** for replication:

```
create procedure balance_limit
as
update accounts
set balance = 0
where balance < 0
```

When Marc executes **balance_limit** on the primary database, Adaptive Server applies the predicated privilege (1) that you defined earlier, and executes the resulting SQL statement, which is equivalent to:

```
update accounts Marc
set balance = 0
where balance < 0
and owner = 'Marc'
```

Replication Server replicates the execution of **balance_limit** to the replicate database, using the maintenance user for the procedure execution on the `accounts` table. Since the maintenance user does not have any predicated privileges, the executed SQL statement at the replicate database is equivalent to:

```
update accounts Marc
set balance = 0
where balance < 0
```

In this example, the SQL statement executed at the primary database is different from the statement executed at the replicate database, resulting in an inconsistency between replicate and primary data.

Granular Permissions

Configure your replication system to ensure replication support for Adaptive Server granular permissions.

Adaptive Server granular permissions enable you to grant system privileges that allow you to construct site-specific roles with privileges to match your requirements, and restrict system administrators and database owners from accessing user data. See *Using Granular Permissions* in the *Adaptive Server Enterprise Security Administration Guide*.

Replication Server supports the replication of the Adaptive Server **sp_restore_system_role** system procedure which you can use to restore a system-defined role to its default privilege condition after applying granular permissions to the role.

With granular permissions in databases within a replication system, configure the replication system to ensure replication support if you:

- Use **rs_init** to:
 - Configure user database maintenance user IDs and RSSD primary user IDs.
 - Add a database to the replication system
 - Upgrade user databases
- Add an Adaptive Server master database as a replicate database into the replication system.
- Use a database replication definition to replicate DDL statements.

Granular Permissions and Replication Server Configuration

You must manually create the Replication Server maintenance user IDs for user databases and RSSD on Adaptive Server.

If you enable granular permissions for Adaptive Server primary and replicate user databases that are part of a replication system, and the user ID that you use to log in to Replication Server and run either the **rs_init** interactive mode or **rs_init** resources file does not possess the `sso_role`, **rs_init** cannot automatically create the:

- Maintenance user ID for the primary and replicate user databases
- Maintenance and primary user IDs for the RSSD if the RSSD resides on Adaptive Server

Instead, after installing Replication Server and before you use **rs_init** in interactive mode or an **rs_init** resource file, you must manually create the user IDs for the primary and replicate user databases and RSSD, and manually grant the relevant permissions that the user IDs require for replication. With the proper permissions, you can use **rs_init** to successfully install RSSD on Adaptive Server and add user databases to the replication system.

Creating and Enabling the Maintenance User Role

Create the maintenance user role and grant privileges that allow the role to apply operations to the user database the maintenance user is going to manage.

1. Start **isql** as the user with system security officer privileges on the Adaptive Server with the master database:

```
isql -Usso_user -Ppassword -Sdata_server_name
```

2. Create the maintenance user role in the Adaptive Server master database and grant the permissions that apply to all operations:

```
use master
go
create role rs_maint_user_role
go
```

3. Grant proxy authorization to the maintenance user role:

```
grant set session authorization to rs_maint_user_role
go
```

4. Grant database-wide privileges to allow the maintenance user acting as the database owner, to execute operations on the user database that the maintenance user is going to manage:

```
use database_name
go
grant manage database permission to dbo
go
```

See the *Privileges Managed by manage database permissions Privilege* table in *Adaptive Server Enterprise Security Administration Guide > Using Granular Permissions > Permission Management > manage database permissions Privilege* for a list of the database-wide privileges.

Granting Required Permissions to Maintenance User Role

Grant to the maintenance user role, Adaptive Server permissions for specific operations.

Prerequisites

Create the maintenance user role.

Task

1. Start **isql** as the user with system administrator privileges on the Adaptive Server that the maintenance user logs into:
`isql -Usa_user -Ppassword -Sdata_server_name`
2. Grant the Adaptive Server permissions to the `rs_maint_user_role` to allow the Replication Server maintenance user the correct privileges to access and update the `database_name` database:

```
use database_name
go
grant delete any table to rs_maint_user_role
go
grant identity_insert any table to rs_maint_user_role
go
grant identity_update any table to rs_maint_user_role
go
grant insert any table to rs_maint_user_role
go
grant select any table to rs_maint_user_role
go
grant truncate any table to rs_maint_user_role
go
grant update any table to rs_maint_user_role
go
grant execute any function to rs_maint_user_role
go
grant execute any procedure to rs_maint_user_role
go
```

Permissions Required for Replication Server Maintenance User

After you install Replication Server and before you use **rs_init** for configuring your replication system, assign the appropriate Adaptive Server permissions to the Replication Server maintenance user.

Table 9. Adaptive Server Permissions Required for Replication Server Maintenance User

Permission	Operations the Permission Authorizes
replication_role	Manage replication operations
delete any table	<ul style="list-style-type: none"> • delete table • lock table
execute any procedure	execute <i>procedure_name</i>
execute any function	execute <i>sql_function_name</i>

Permission	Operations the Permission Authorizes
identity_insert any table	set identity_insert {on off}
identity_update any table	set identity_update {on off}
insert any table	<ul style="list-style-type: none"> • insert • lock table
select any table	select
truncate any table	truncate table
update any table	<ul style="list-style-type: none"> • update • lock table • writetext

Creating the Maintenance User and Assigning Roles

Create the maintenance user and assign the relevant roles to the user.

1. Start **isql** as the user with system security officer privileges on the Adaptive Server with the user database that the maintenance user is going to manage:
`isql -U sso_user -P $password$ -S $data_server_name$`

2. Create the maintenance user login ID:

```
create login  $maintenance\_user\_name$  with password
 $maintenance\_user\_password$ 
go
```

3. Grant replication_role to the maintenance user ID:

```
sp_role "grant", replication_role,  $maintenance\_user\_name$ 
go
```

You see:

```
Authorization updated.
(return status = 0)
```

4. Grant rs_maint_user_role to the maintenance user ID:

```
sp_role "grant", rs_maint_user_role,  $maintenance\_user\_name$ 
go
```

You see:

```
Authorization updated.
(return status = 0)
```

5. Automatically activate rs_maint_user_role when the maintenance user logs in to the user database:

Manage RepAgent and Support Adaptive Server

```
alter login maintenance_user_name add auto activated roles  
rs_maint_user_role  
go
```

6. Assign `dbo`, the user database owner, as an alias for the maintenance user to allow the maintenance user to manage the database with all the privileges of `dbo`:

```
sp_addalias maintenance_user_name, dbo  
go
```

Note: Do not execute `sp_addalias` if you are configuring master database replication.

You see:

```
Alias user added.  
(return status = 0)
```

Granular Permissions and Adding a User Database

If you enable granular permissions, and then manually configure the maintenance user permissions, you see an error message when you use the `rs_init` interactive mode to add a database to the replication system.

```
Sending the following SQL command to the server:  
'sp_role "grant", replication_role, <maintenance user name>'.  
SQL Server message: msg 10331, level 14, state 2  
WARNING: "Permission denied, database master, owner dbo.  
You need the following permission(s) to run this command: MANAGE  
ROLES."
```

Since you have manually granted the required permissions to the maintenance user IDs, ignore the error message and press the Enter key to continue.

Granular Permissions and User Database Upgrade

If you use `rs_init` to upgrade Adaptive Server user databases, you see an error message.

If you enable granular permissions for Adaptive Server and then use `rs_init` to upgrade Adaptive Server user databases, `rs_init` attempts to grant the `replication_role` permission to the user ID that you used to execute `rs_init` and also to the maintenance user ID for the user databases. However, `rs_init` fails to grant the `replication_role` permission because since `dbo`, the master database owner, does not have the `manage roles` permission, and you see an error message such as:

```
SQL Server message: msg 10331, level 14, state 2  
WARNING: "Permission denied, database master, owner dbo.  
You need the following permission(s) to run this command: MANAGE  
ROLES."
```

Ignore the error message and press the Enter key to continue because the user database upgrade can still proceed.

Granular Permissions and Master Database Replication

Remove the maintenance user ID alias relationship to the dbo master database owner before you add the master database as a replicate database.

If you enable granular permissions for Adaptive Server and use **rs_init** to try to add an Adaptive Server master database to the replication system as a replicate database, you may see error messages such as:

```
2012/09/17 00:47:47 SQL Server message: msg 11105, level 11, state 1
2012/09/17 00:47:47 WARNING: "No such user/role 'xxxxxx' exists.
"2012/09/17 00:47:47 Unable to execute query 'grant execute on
rs_update_lastcommit to xxxxxx' against server 'xxxxxx'.
```

The attempt to use **rs_init** to add the master database fails, because in the master database, the maintenance user ID is currently an alias of dbo. However, you cannot specify the maintenance user ID as the alias of dbo for the master database.

Use the **sp_dropalias** Adaptive Server system procedure to remove the alias at the master database:

```
sp_dropalias loginame
```

Granular Permissions and DDL Replication

Enable **dsi_replication_ddl** to replicate DDL statements.

If you enable granular permissions for Adaptive Server and you use a database replication definition to replicate DDL statements, and the user who executes the DDL statement does not have the **execute** permission for the **rs_update_last_commit** stored procedure on the replicate database, the DDL statement fails with an error message such as:

```
Message from server: Message: 10330, State 1, Severity 14 --
'EXECUTE permission denied on object rs_update_lastcommit, database
rdb1, owner dbo
```

If you enable granular permissions, set **dsi_replication_ddl** on for the connection to the replicate database to allow Replication Server to use the maintenance user to execute **rs_update_last_commit**:

```
alter connection to dataserver.database set dsi_replication_ddl on
```

Stored Procedure Replication and the execute as Clause

Ensure support for stored procedure replication when you create stored procedures with the **execute as {owner | caller}** clause of the Adaptive Server **create procedure** command.

Adaptive Server Stored Procedure Creation

In Adaptive Server, you can create a stored procedure with the **execute as {owner | caller}** clause.

If you create a stored procedure:

- Without an **execute as** clause, Adaptive Server resolves the names of objects inside the stored procedure on behalf of the procedure owner. Adaptive Server evaluates permission checks for DML inside the stored procedure on behalf of the procedure caller, and then executes DDL inside the stored procedure on behalf of the procedure caller.
- With an **execute as owner** clause, Adaptive Server resolves the names of objects inside the stored procedure on behalf of the procedure owner. Adaptive Server evaluates permission checks for DML inside the stored procedure on behalf of the procedure owner, and then executes DDL inside the stored procedure on behalf of the procedure owner.
- With an **execute as caller** clause, Adaptive Server resolves the names of objects inside the stored procedure on behalf of the procedure caller. Adaptive Server evaluates permission checks for DML inside the stored procedure on behalf of the procedure caller, and then executes DDL inside the stored procedure on behalf of the procedure caller.

See *Executing a Procedure with execute as owner or execute as caller* in the *Adaptive Server Enterprise Security Administration Guide*.

Replication Server Function Replication Definitions

In Replication Server, you can use the **create applied function replication definition** or **create request function replication definition** commands to create function replication definitions that describe stored procedures that you want to replicate..

The difference between the applied function replication definition and the request function replication definition is that the function that Replication Server replicates through an applied function replication definition is executed at the replicate site by the maintenance user while the function that Replication Server replicates through a request function replication definition is executed at the replicate site by the same user who executes the primary function at the primary site. See *Manage Replicated Functions* in the *Replication Server Administration Guide Volume 1*, and **create applied function replication definition** and **create request function replication definition** in the *Replication Server Reference Manual*

To replicate a stored procedure in a non-warm standby or non-MSA environment, you must create a replication definition and a corresponding subscription. Replication of a stored procedure in a warm standby or MSA environment does not require a replication definition. Only one function replication definition can exist for a specific replicated stored procedure.

Ensuring Successful Stored Procedure Replication

If you use the **execute as {owner | caller}** clause to create procedures, the identity of the caller at the primary database is not always preserved at the replicate database.

If the identities differ, stored procedure replication may fail because:

- The DDL permissions at the replicate database are insufficient for Adaptive Sever to execute the stored procedure DDL.

- The replicate database cannot resolve the names of objects inside the stored procedure which results in the database accessing the wrong objects or not finding any objects.
- The DML cannot pass the replicate database permission evaluation checks.

For the successful replication of stored procedures:

- Carefully evaluate object name resolution or whenever possible, use fully qualified object names when you create stored procedures with the **execute as {owner | caller}** clause.
- Grant the relevant permissions to the:
 - Maintenance user if you are using an applied function replication definition.
 - Replicate procedure owner if you are using a request function replication definition.

Precomputed Result Sets

Replication Server supports the replication of precomputed result sets DDL commands between primary and replicate Adaptive Server databases that support precomputed result sets.

Adaptive Server 15.7 ESD #2 and later supports precomputed result sets. See *Precomputed Result Sets* in the *Adaptive Server Enterprise New Features Guide version 15.7 ESD #2*.

In Adaptive Server, you can create a precomputed result set on any query expression, together with a set of policies to maintain the precomputed result set. Replication Server supports the replication of precomputed result set data definition language (DDL) commands between Adaptive Server databases in all environments, including multisite availability (MSA) and warm standby environments. However, you cannot mark precomputed result sets for replication, and the **update**, **delete**, **insert**, and **select into** data manipulation language (DML) commands cannot directly update the precomputed results set. This is because unlike a regular table, Replication Server does not replicate any maintenance changes occurring on data stored in a precomputed result set, whether the changes are:

- Initiated from a precomputed result set DDL, although the DDL can also be replicated, or
- From part of the base table update transaction initiated from the **immediate refresh** policy.

Replication Server only supports the replication of the Adaptive Server precomputed result set DDL commands:

- **alter {precomputed result set | materialized view}**
- **create {precomputed result set | materialized view}**
- **drop {precomputed result set | materialized view}**
- **refresh {precomputed result set | materialized view}**
- **truncate {precomputed result set | materialized view}**

Precomputed result sets are also called materialized views. See *Commands* in the *Adaptive Server Reference Manual: Commands*.

Adaptive Server requires four session-level **set** options to have the relevant settings before you can create and use precomputed result sets:

- **set ansinull on**
- **set arithabort on**
- **set arithignore off**
- **set string_rtruncation on**

To replicate the precomputed result set DDL commands, Replication Server includes the **rs_ddlsession_setting** system function to automatically set the Adaptive Server session-level options whenever the DSI thread processes these DDL commands, and therefore allow Replication Server to apply the commands to the replicate Adaptive Server database. To avoid affecting DML replication, Replication Server provides the **rs_ddlsession_resetting** system function to automatically reset the session-level options to their default values after processing the precomputed result set DDL commands:

- **set ansinull off**
- **set arithabort on**
- **set arithignore off**
- **set string_rtruncation off**

Remember: You do not have to execute the functions manually.

Tip: The **rs_ddlsession_resetting** function resets the options to the default values overwriting any values you previously set. You can create customized a function string for **rs_ddlsession_resetting** to ensure that the values revert to what you previously set.

See the descriptions of **rs_ddlsession_setting** and **rs_ddlsession_resetting** in the *Replication Server Reference Manual*.

Earlier versions of Replication Server that do not include the two functions can replicate precomputed result set DDL commands as long as both primary and replicate Adaptive Server databases support precomputed result sets and as long you can apply the relevant settings for the Adaptive Server session-level **set** options. To ensure that precomputed result set DDL commands can replicate even if your version of Replication Server version does not include the system functions, you can create a procedure in the default database that automatically configures the relevant **set** options when the original user for the replicate database logs into the database.

Enabling Precomputed Result Sets Replication for Earlier Versions of Replication Server

Create a procedure in the default database to automatically apply the relevant values for the session-level **set** options and ensure that precomputed result set DDL commands successfully replicate even if the Replication Server version does not include the **rs_ddlsession_setting** and **rs_ddlsession_resetting** system functions.

You must create the procedure in the default Adaptive Server database. See *Adaptive Server Enterprise Transact-SQL Users Guide > SQL Building Blocks > isql utility > Default*

Databases. If you do not specify a database, the default is the master database. You can include the procedure in the login script of the original user or in the login script of the maintenance user if **dsi_replication_ddl** is on. Once the user logs in to the database to apply transactions, the procedure applies the relevant values for the session-level options and all operations of the user are impacted by the session-level **set** options..

1. Create a procedure that applies the appropriate values to the Adaptive Server session-level set options that Adaptive Server requires for support of precomputed result sets.

To create the **pcrs_proc** procedure at the `rdb1` replicate database, enter:

```
create procedure pcrs_maint_login_proc
as
set ansinull on
set arithabort on
set arithignore off
set string_rtruncation on
go
grant all on pcrs_proc to public
go
```

2. Include the procedure that enables precomputed result set in the login script for the original user of the default replicate database.

To include **pcrs_proc** in the login script for the `rdb1_user1` database user who logs into the `rdb1` database, enter:

```
sp_modifylogin rdb1_user1, 'login script', pcrs_proc
```

If you use the maintenance user to define the login script, DML replication may not proceed properly.

Transfer of Database Object Ownership

Replication Server supports the transfer of database object ownership in Adaptive Server.

For employee life cycle management in Adaptive Server, database administrators, system security officers, or database owners can manage the assignment of database objects due to employee changes by transferring the ownership of database objects using the **alter... modify owner** Adaptive Server command. In addition, database administrators can separate the creation of objects from the ownership of objects by using **alter... modify owner**. See *Adaptive Server Enterprise Security Administration Guide > Managing User Permissions > Changing database object ownership*.

Replication Server Support for Transferring Object Ownership

Replication Server replicates the DDL changes generated by **alter... modify owner** so that the transfers in object ownership at the primary Adaptive Server database are applied to the replicate Adaptive Server database. However, if there is a change in object ownership for an object qualified with an owner, in a replication definition where the owner name is required, Replication Server does not automatically alter the replication definition to reflect the change.

Suppose you create a replication definition for the `authors` table with Mario as the owner —`mario.authors`, and subsequently the database administrator transfers the table owner to Angela. As a result, Replication Server cannot replicate any **insert**, **delete**, or **update** operations to the replicate `authors` table because Replication Server receives these operations for replication to `angela.authors` but the replication definition still has `mario.authors`.

Coordination of Object Ownership Transfer and Replication Definition Changes

To ensure that data replicates continuously and correctly, coordinate any changes to object ownership with corresponding changes to affected replicated definitions and database replication definitions.

Executing **`rs_send_repserver_cmd`** at the primary database allows you to order the change to the replication definition to be always after executing **`alter... modify owner`**. It is more difficult to coordinate the transfer of object ownership with the modifications to the replication definition or database replication definition if you execute **`alter replication definition`** or **`alter database replication definition`** at the Replication Server.

At the primary database, immediately after using **`alter... modify owner`**, execute the **`rs_send_repserver_cmd`** stored procedure that in turn executes the **`alter replication definition`** or **`alter database replication definition`** command containing the **`alter owner from current_owner to new_owner`** clause to change the object owner in the replication definition or database replication definition:

- **`alter replication definition`** –

```
exec rs_send_repserver_cmd 'alter replication
definition replication_definition_name
alter [primary] owner from current_owner to new_owner'
```

- **`alter database replication definition`** –

```
exec rs_send_repserver_cmd 'alter database replication
definition database_replication_definition_name
alter owner from current_owner to new_owner [for tablename]'
```

Omit the **`for tablename`** option if you want to change the owners for all tables in the database replication definition .

Altering the owner in the replication definition creates a new version of the replication definition. If there are already multiple versions of the replication definition, Replication Server only modifies the current version to ensure that any data in the inbound queue replicates with the proper replication definition version. This allows for the transfer of table ownership to occur without having to drain the queues.

Wildcards for Changing Multiple Replication Definitions

If you use **`alter... modify owner`** to transfer the ownership of multiple tables, you can use the asterisk "*" wildcard character to replace the replication definition name in **`alter replication definition`** to modify the owner in multiple replication definitions affected by the ownership transfer, where *old_owner* is the same for all the replication definitions:

```
exec rs_send_repserver_cmd 'alter replication
definition *
alter owner with primary at data_server.database
from old_owner to new_owner'
```

Check for Affected Replication Definitions

Before you use the wildcard to change multiple replication definitions, execute the **rs_helpreptable** stored procedure in the RSSD with a wildcard "*" to list all the replication definitions at the primary database that have the same *current_owner* value:

```
rs_helpreptable primary_database, current_owner, '*'
```

Scenario for Coordinating Replication Definition Changes with Transfers to Object Ownership

You must immediately follow any change to table ownership at the primary database with the corresponding changes to affected replication definitions to ensure replication proceeds correctly.

This example shows how to coordinate transfers to table ownership at an Adaptive Server primary database with the corresponding changes to affected table replication definitions.

1. Create the **authors_repdef** replication definition for the **authors** table at the **pubs2** database in the **NY_DS** data server with **Mario** as the table owner. At the primary Replication Server enter:

```
create replication definition authors_repdef
with primary at NY_DS.pubs2
with primary table named mario.authors
```

2. Mark the table for replication and specify that the owner of the primary table must match the replicate table owner for replication to proceed. At **pubs2** primary database enter:

```
sp_setreptable mario.authors, 'true', owner_on
```

3. The database administrator or owner transfers ownership of the **authors** table from **Mario** to **Angela** at **pubs2**:

```
alter table mario.authors
modify owner angela
```

4. Verify if **authors_repdef** is the only replication definition affected by the change in table ownership from **Mario**. At the Replication Server enter:

```
rs_helpreptable pubs2, mario, *
```

If there is more than one replication definition affected by the change in table ownership, you can modify all replication definitions in one command using the wildcard character.

5. You must change **authors_repdef** immediately to maintain seamless replication and data integrity at the replicate database. At **pubs2** enter:

```
exec rs_send_repserver_cmd 'alter replication
definition authors_repdef
alter primary owner from mario to angela'
```

Conditions and Limitations for Replication Server Support for Transfer Object Ownership

There several conditions and limitations for replication of objects affected by a transfer of object ownership.

- Owner status – if you use **sp_setreptable** to set the owner status for a primary table to:
 - **owner_on** – and subsequently there is a change in table owner, you must immediately make the corresponding change in affected replication definitions.
 - **owner_off** – and subsequently there is a change in table owner, do not make the corresponding change in affected replication definitions. Otherwise, the replication definition filters out all transactions for the table, resulting in no data replication.
- DDL replication – you must ensure that Replication Server can replicate DDL commands before you execute **alter... modify owner** and **rs_send_repserver_cmd**.

Suppose you set **owner_on** but you do not enable DDL replication by setting **sp_setreptostandby** to **none** for warm standby systems, or by setting **set replication** to **off** for non-warm standby replication. Assuming that you do not make corresponding changes to the table replication definition, any change you make with **alter...modify owner** at the primary database does not replicate. As a result, the table owner of the affected table at the primary database is different from the owner of the replicate table. Therefore, data cannot replicate to the replicate table.

Suppose you also alter the table replication definition to change the owner at the:

- Primary and replicate tables with:

```
alter replication definition replication_definition_name
alter owner from old_owner to new_owner
```

Replication Server forms SQL statements to replicate to *new_owner.table_name* which does not exist since the actual replicate table owner name was not changed. This causes an error in the replication process that suspends the DSI thread and the connection to the replicate database. You can then modify the table owner in the replicate database and resume the connection. Replication continues normally without data loss.

- Primary table with:

```
alter replication definition replication_definition_name
alter primary owner from old_owner to new_owner
```

Replication Server receives data for replication from the primary table that is owner-qualified as *new_owner.table_name* and replicates these changes to the *old_owner.table_name* replicate table.

Warning! Data replication loss can result if you have an incorrect assumption about the status of DDL replication in your replication environment and alter your replication definitions incorrectly. Understand your replication environment before you make changes to replication definitions affected by changes to object ownership.

- Different table owners – Replication Server does not map tables owners from a primary owner to a different replicate owner
- Different table names – when you create replication definitions, you can specify table names at the replicate database that are different from the table names at the primary database:

```
create replication definition replication_definition_name
with primary at data_server.databasew
with primary table named table_owner1.pri_table
with replicate table named table_owner1.rep_table
```

If there is a requirement for a transfer of object ownership, execute **alter...modify owner** at the primary database and then at the replicate database. Immediately after executing **alter...modify owner**, execute the **rs_send_repserver_cmd** stored procedure at the primary database or change the replication definition to modify the table names in the affected replication definition.

However, replication fails if there is any data in the queue where tables have the old owner. To ensure that there is no data loss during replication, either:

- Drain the queue before changing the owner of the replicate table, or
- After changing the table owner, allow the DSI to suspend if the primary table receives new data and then execute **rs_send_repserver_cmd**
- Same table with multiple projections of the same replication definition – you can create multiple replication definitions for the same primary table and customize each one, called a projection, so each replication definition projection can be subscribed to by a replicate table whose characteristics are different from those of the primary table and other replicate tables. The other projections can replicate the same table using a subset of the commands or to a different replicate table.

After a transfer of object ownership at the primary table, if you use

rs_send_repserver_cmd with **alter replication definition** or **alter replication definition** by itself to modify the:

- Primary table owner – Replication Server only modifies the primary table owner of the replication definition you specified.
- Primary and replicate table owner – Replication Server modifies both the primary and replicate table owner of the replication definition you specified.

Replication Server modifies all replication definitions and projections if you use the wildcard "*" instead of replication definition name. If modifying a replication definition causes the replication definition and a projection to be identical, Replication Server does not proceed with the modification and returns an error instead.

- Mixed-version support – Replication Server does not allow you to change table ownership in a replication definition or database replication definition if there is a route to a replicate Replication Server with a route version earlier than 15.5
- Impact on HVAR and MSA – high-volume adaptive replication (HVAR) and multisite availability (MSA) cache replication definitions during processing. If you transfer object ownership for a table and modify the affected replication definitions, you must invalidate the cached replication definition by disabling HVAR for the table or by stopping MSA.

Backlinking Pointers and Shrinking Databases

Replication Server supports backlinking pointers and shrinking databases in Adaptive Server.

Replication of the Transact-SQL **writetext** command requires access to the data row pointing to the text page where the database stores the LOB data. To allow access to this data row, Adaptive Server uses either a backlinking pointer in the first text page or indexes created for replication. The process of creating indexes at the column, table or database level requires an intensive operation to provide the information to support replication.

With an Adaptive Server version 15.7 SP100 and later database that you did not upgrade from an earlier version, **sp_reptostandby** takes effect immediately because by default, Adaptive Server creates and maintains LOB backlinking pointers to the database. Therefore, setting up replication for a table does not require the creation of indexes. Adaptive Server ignores the **use_index** parameter of **sp_reptostandby**, **sp_setrepcol**, and **sp_setreptable** if the information needed to replicate LOB columns is already available in the form of backlinking pointers.

However, if you have upgraded from, or are using a database that you created with a version of Adaptive Server earlier than 15.7 SP100, setting up replication may take a longer time due to the creation of indexes. To reduce processing time, run **dbcc shrinkdb_setup** at the relevant level—column, table, or database, to create backlinking pointers and to ensure the backlinking status is up to date.

dbcc shrinkdb_setup marks as suspect, replication indexes of columns, tables, or databases that you previously marked with **use_index**. You can use **dbcc reindex** to drop indexes for these objects because these indexes are not needed after the execution of **dbcc shrinkdb_setup**.

See *Shrinking Databases* in the *Adaptive Server Enterprise 15.7 SP100 New Features Guide*.

Performance Enhancements for Adaptive Server Replication

Replication Server provides several features that enhance performance for Adaptive Server replication.

- SQL Statement Replication – Replication Server supports SQL statement replication in Adaptive Server which complements log-based replication and addresses performance degradation caused by batch jobs. Sybase recommends that you use SQL statement replication when:
 - DML (data manipulation language) statements affect a large number of rows on replicated tables.

- You have difficulty altering the underlying application to enable stored procedure replication.
- High Volume Adaptive Replication – Replication Server includes High Volume Adaptive Replication (HVAR) to enhance performance for replication to Adaptive Server. HVAR compiles and combines a larger number of transactions into a group, improving bulk operation processing. Therefore, replication throughput and performance also improves. HVAR is especially useful for creating online transaction processing (OLTP) archiving and reporting systems where the replicate databases have the same schemas as the primary databases.

See *Replication Server Administration Guide Volume 2 > Performance Tuning > SQL Statement Replication* and *Replication Server Administration Guide Volume 2 > Performance Tuning > Advanced Services Option > High Volume Adaptive Replication*.

In-memory and Relaxed-Durability Databases

Replication Server supports the in-memory and relaxed-durability database Adaptive Server feature.

In-memory databases (IMDB) reside entirely in cache and do not use disk storage for data or logs, and therefore do not require disk I/O. This results in potentially greater performance than a traditional disk-resident database (DRDB), amongst other advantages. Since an in-memory database exists only in cache, you cannot recover the database if the supporting host is shut down or the database fails.

With relaxed-durability databases, Adaptive Server extends the performance benefits of an in-memory database to disk-resident databases. Disk-resident databases perform writes to disk, and ensure that the transactional properties of atomicity, consistency, integrity, and durability, known as the ACID properties, are maintained. A traditional disk-resident database operates at full durability to guarantee transactional recovery from a server failure. Relaxed-durability databases trade the full durability of committed transactions for enhanced runtime performance for transactional workloads. A relaxed-durability database created with the **no_recovery** level is similar to an in-memory database: you cannot recover data or logs if the server terminates or is shut down. You can also create a relaxed-durability database with the **at_shutdown** level where transactions are written to disk if there is a proper shut down of the database.

See *Adaptive Server Enterprise In-Memory Database Users Guide*.

Replication Server Support

Replication Server supports as the primary and replicate database:

- In-memory databases
- Relaxed-durability databases set with durability at **no_recovery**

For convenience, relaxed-durability databases refers to relaxed-durability databases with durability set to **no_recovery**.

You can initialize an in-memory and a relaxed-durability database as a new primary or replicate database by obtaining data, object schema, and configuration information from one of:

- A template database that retains basic information.
- A database dump from another database and then load the dump to the target in-memory database or relaxed-durability database.

The dump source database can be another in-memory database, relaxed durability database, or a traditional disk-resident database.

Note: Replicating to or from an in-memory database may not be faster than replicating to or from a relaxed durability database. DML on in-memory databases depends on several factors.

In addition, there is no difference in performance when replicating to or from an in-memory or relaxed durability database compared to replication between primary and replicate databases which are traditional full durability disk-resident databases.

In-Memory Databases as Primary Databases for Replication

You can configure in-memory and relaxed-durability databases as primary databases in a replication system.

Using a Template Database to Set Up Replication for a Primary In-Memory Database

You can use the same disk-resident database as a template to initialize multiple in-memory or relaxed-durability databases as primary databases. The primary in-memory or relaxed-durability database inherits the configuration of the template database.

Note: After a shutdown or termination and a subsequent restart of Adaptive Server, Adaptive Server recreates the in-memory or relaxed-durability database automatically from the template.

1. Create the template database. The template database uses the name of the database with the inbound connection from Replication Server which is usually the primary database name.

For example, to create the template database named `ny_db` in the `NY_DS` data server:

```
create database ny_db on publicdev=10 log on publicdevlog=10
go
```

2. Use `rs_init` to add the primary and replicate databases to the replication system.
3. Stop RepAgent on the template database:

```
sp_stop_rep_agent ny_db
go
```

4. Suspend the connection from Replication Server to the database.
5. Rename the template database to `templatel`:

```
use master
go
sp_dboption ny_db, single, true
go
sp_renamedb ny_db, templatel
go
sp_dboption templatel, single, false
go
```

6. Create the in-memory or relaxed durability database with durability set to **no_recovery** using the template created in step 1:

```
create inmemory database ny_db
use templatel as template
on imdb_cache_dev = '50' log on imdb_cache_dev_log='50'
with DURABILITY=NO_RECOVERY
go
```

7. Configure RepAgent to start automatically after Adaptive Server shuts down and restarts:

```
use templatel
go
sp_config_rep_agent templatel, 'auto start', true
```

8. Resume the connection from Replication Server to the database.

Using a Database Dump to Set Up Replication for a Primary In-Memory Database

You can set up replication if you use a dump from another database to initialize the in-memory or relaxed-durability as a primary database. The primary in-memory or relaxed-durability database inherits the configuration of the database from which you obtained the dump.

1. Create the in-memory or relaxed-durability database:

```
create inmemory database ny_db
on imdb_cache_dev2 = '50' log on imdb_cache_dev_log2='50'
with DURABILITY=NO_RECOVERY
go
```

2. Use **rs_init** to add the primary in-memory database to the replication system.
3. Obtain a dump from the database that you want to use load the new in-memory database you created.
4. Load the in-memory database from the database dump.

For example:

```
use master
go
sp_dboption ny_db, single, true
go
load database ny_db from '/remote/Based_on_loaddb/IMDB.dump'
go
online database ny_db
go
```

Manage RepAgent and Support Adaptive Server

```
sp_dboption ny_db, single, false
go
```

5. Start RepAgent on the database:

```
sp_start_rep_agent ny_db
go
```

6. Optionally, resume the DSI connection from Replication Server to the primary database.

Shutdown or Termination of an In-Memory Database

After a shutdown or termination and a subsequent restart of Adaptive Server, Adaptive Server recreates the in-memory or relaxed-durability database automatically from the template.

When Replication Server resumes the connection to the in-memory or relaxed-durability database that you created, Replication Server may reapply commands, as the information used by Replication Server to detect the latest command that was applied is lost when you restart Adaptive Server.

In-Memory Databases as Replicate Databases for Replication

You can configure in-memory and relaxed-durability databases as replicate databases in a replication system.

Using a Template Database to set up Replication for a Replicate In-Memory Database

You can use the same disk-resident database as a template to initialize multiple in-memory or relaxed-durability databases.

Note: After a shutdown or termination and a subsequent restart of Adaptive Server, Adaptive Server recreates the in-memory or relaxed-durability database automatically from the template. When Replication Server resumes the connection to the replicate in-memory or relaxed-durability database that you created, Replication Server may reapply commands, as the information used by Replication Server to detect the latest command that was applied is lost when you restart Adaptive Server.

1. Create the template database. The template database uses the name of the database with the outbound connection to Replication Server which is usually the replicate database name.

```
create database tokyo_db on publicdev=10 log on publicdevlog=10
go
```

2. Use **rs_init** to add the replicate database to the replication system.
3. Suspend the DSI thread to the template database by stopping RepAgent on the template database. For example:

```
suspend connection to TOKYO_DS.tokyo_db
```

4. Rename the template database to **template1**:

```
use master
go
sp_dboption tokyo_db, single, true
```

```
go
sp_renamedb tokyo_db, template1
go
sp_dboption template1, single, false
go
```

5. Create the in-memory or relaxed durability database with durability set to **no_recovery** using the template created in step 1:

```
create inmemory database tokyo_db
use template1 as template
on imdb_cache_dev = '50' log on imdb_cache_dev_log='50'
with DURABILITY=NO_RECOVERY
go
```

6. Connect to Replication Server and resume the connection to the replicate database:

```
resume connection to TOKYO_DS.tokyo_db
```

Using a Database Dump to set up Replication for a Replicate In-Memory Database

You can set up replication if you use a dump from another database to initialize the in-memory or relaxed-durability as a replicate database. The replicate in-memory or relaxed-durability database inherits the configuration of the database from which you obtained the dump.

1. Create the in-memory or relaxed-durability database:

```
create inmemory database tokyo_db
on imdb_cache_dev2 = '50' log on imdb_cache_dev_log2='50'
with DURABILITY=NO_RECOVERY
go
```

2. Create the objects such as, tables and stored procedures, users, and permissions required to receive replicate data, or you can load a database dump.
3. Use **rs_init** to create the Replication Server connection to the in-memory or relaxed-durability database.
4. Perform a dump to save the current state of the in-memory or relaxed-durability database:

- a) Suspend the connection to the in-memory or relaxed-durability database:

```
suspend connection to RDS.imdb1
go
```

- b) Obtain a database dump of the in-memory or relaxed-durability database:

```
dump database imdb1 to '/databases/dump/tokyo_db.dump'
go
```

- c) Resume the connection to the in-memory or relaxed-durability database:

```
resume connection to RDS.imdb1
go
```

Restoration of In-Memory and Relaxed-Durability Databases

Since in-memory and relaxed-durability databases exist only in cache, they lose their object definition, data, and RepAgent configuration once the host data server shuts down or restarts.

You must reinitialize or restore the in-memory or relaxed durability database after the host data server restarts.

You can reinitialize the in-memory or relaxed durability database from the template or you can restore the in-memory or relaxed durability database with a database dump using one of these methods:

- Recreating from a dump
- Database resynchronization
- Bulk materialization

Ensure that you have enough disk space and time to perform a database dump and load, and that the period of time during which Replication Server skips transactions is acceptable. You can estimate the acceptable period of time by monitoring the segments in the outbound queue with **admin who**, **sqm**. See *Replication Server Reference Manual > Replication Server Commands > admin who*

Recreating In-Memory and Relaxed-Durability Databases From a Dump After an Adaptive Server Shutdown

You can restore the data from the dump because the in-memory and relaxed-durability databases are recreated when you restart Adaptive Server.

1. Repopulate the recreated replicate in-memory or relaxed-durability database with a new dump or from any archived dumps taken from the replicate database.

Note: If the dump is not loaded from the dump of the source, there will be missing rows in the replicate tables.

For example, to load the `tokyo_db` database from the original `tokyo_db.source` dump when the host Adaptive Server restarts:

```
use master
go
sp_dboption tokyo_db, single, true
go
load database tokyo_db from '/databases/dump/tokyo_db.dump'
go
online database tokyo_db
go
sp_dboption tokyo_db, single, false
go
```

2. Resume the connection to the recreated replicate in-memory or relaxed-durability database.

Resynchronizing Directly from a Primary Database

Resynchronize an in-memory database from a primary database.

1. Stop replication processing by RepAgent. In Adaptive Server, execute:

```
sp_stop_rep_agent database
```


2. Suspend the Replication Server DSI connection to the replicate database:

```
suspend connection to dataserver.database
```

3. Instruct Replication Server to remove data from the replicate database outbound queue and wait for a resync marker from the primary database RepAgent:

```
resume connection to data_server.database skip to  
resync marker
```

4. Instruct RepAgent to start in resync mode and send a resync marker to Replication Server:

- If the truncation point has not been moved from its original position, in Adaptive Server execute:

```
sp_start_rep_agent database, 'resync'
```

- If the truncation point has been moved from its original position, in Adaptive Server execute:

```
sp_start_rep_agent database, 'resync purge'
```

5. In the Replication Server system log, verify that DSI has received and accepted the resync marker from RepAgent by looking for this message:

```
DSI for data_server.database received and processed  
Resync Database Marker. Waiting for Dump Marker.
```

Note: If you are resynchronizing multiple databases, verify that the DSI connection for each of the databases you want to resynchronize has accepted the resync marker.

6. Obtain a dump of the primary database contents. See *Adaptive Server Enterprise Reference Manual: Commands > Commands > **dump database***. Adaptive Server automatically generates a dump database marker.
7. Verify that Replication Server has processed the dump database marker by looking for this message in the Replication Server system log:

```
DSI for data_server.database received and processed  
Dump Marker. DSI is now suspended. Resume after database has been  
reloaded.
```

When Replication Server receives the dump marker, the DSI connection automatically suspends.

8. Apply the dump of the primary database to the replicate database. See *Adaptive Server Enterprise Reference Manual: Commands > Commands > **load database***.
9. After you apply the dump to the replicate database, resume DSI:

```
resume connection to data_server.database
```

Resynchronizing a Replicate In-Memory or Relaxed-Durability Database with Bulk Materialization

You can use one of two bulk materialization methods to restore an in-memory or relaxed-durability database.

Prerequisites

Before you start bulk materialization, verify that the replication definitions and subscriptions exist.

Task

1. To quickly empty the inbound and outbound queues, deactivate the subscriptions that have the in-memory or relaxed-durability database:

```
deactivate subscription subscription_name
for {table_repdef_name | func_repdef_name | {publication pub_name |
database replication definition db_repdef_name}
with primary at dataserver.database}
with replicate at dataserver.database
go
```

After you deactivate the subscriptions, Replication Server does not propagate all the transactions in the inbound queue to the outbound queue of the in-memory or relaxed-durability database.

In contrast, when you drop a subscription, all the committed transactions that have been written into the inbound queue are distributed downstream of Replication Server. You can deactivate a subscription even if the DSI is not running because the deactivation only happens at the primary site. When the deactivate marker arrives at the outbound queue, you can see this entry in the Replication Server log:

```
The deactivate marker for subscription subscription_name
arrives at outbound queue: data_server_name.database_name.
```

After the deactivate marker arrives at the outbound queue, use **sysadmin sqm_purge_queue** to purge the outbound queue at the replicate site to quickly empty the outbound queue. See *Replication Server Reference Manual* > *Replication Server Commands* > **sysadmin sqm_purge_queue**.

2. Execute **check subscription** at both the primary and replicate Replication Servers to verify that the subscription status is DEFINED at the primary Replication Server and VALID at the Replication Server.
3. Use the "Simulate Atomic Materialization" or "Simulate Nonatomic Materialization" bulk materialization methods described in *Replication Server Administration Guide Volume 1* > *Manage Subscriptions* > *Subscription Materialization Methods* > *Bulk Materialization*, to build the in-memory or relaxed-durability database. If you use:

- Simulate atomic materialization — execute steps 4 to 9
- Simulate nonatomic materialization — execute steps 4 to 13

Enable Autocorrection

Before you resume the connection to the replicate in-memory or relaxed-durability database, you must enable autocorrection for the replication definitions used for subscriptions to the replicate database, in order to convert any update or insert into a delete and insert pair.

Enabling autocorrection applies to replicate databases created using templates or dumps. Autocorrection allows Replication Server to continue replicating messages in the Replication Server queues even if the Adaptive Server hosting a replicate in-memory or relaxed-durability database shuts down or terminates.

Minimal DML Logging and Replication

Replication Server supports minimal DML logging in an in-memory or relaxed durability database acting as the replicate database.

To optimize the log records that are flushed to the transaction log on disk, Adaptive Server can perform minimal to no logging when executing some data manipulation language commands (DMLs)—**insert**, **update**, **delete**, and slow **bcp**—on all types of low-durability databases, such as in-memory databases and relaxed-durability databases set with durability of **at_shutdown** or **no_recovery**. You can perform minimal logging for DMLs on a per-database, per-table, and session-specific basis. See *Adaptive Server Enterprise In-Memory Database Users Guide > Minimally-logged DML*.

Note: Minimal DML logging session-level settings take precedence over database-level settings and table-level settings.

As replication uses full logging, replication and the minimal data manipulation language (DML) logging feature in Adaptive Server 15.5 are incompatible at the same level, such as database-level or table-level. However, you can take advantage of the performance enhancements from minimal logging on some tables while allowing replication on others as minimal DML logging and replication can coexist at different levels.

For example, if you set replication and minimal DML logging at the same level, such as table-level, setting replication status fails and an error message appears as described in these scenarios:

- If you create a database to use minimal DML logging:
 - And you mark the database for replication using **sp_reptostandby**, the attempt fails and you see:

```
Cannot set replication for database database_name
as it is minimally logged. Use ALTER DATABASE to
set full DML logging and try again.
```

- And you mark tables and stored procedures for replication to replicate a subset of tables, and mark a table in the database using minimally DML logging, you see this warning:

```
Warning: database_name is using minimal logging.  
Replicated objects will continue to use full DML  
logging.
```

- If a database is using full logging and you mark it for replication using **sp_reptostandby**, and you alter the database to set minimal DML logging, the attempt fails and you see:

```
Cannot alter database database_name to set minimal  
logging because this database is marked for  
replication. Remove replication and try again.
```

- If a database is using full logging and has objects marked for replication, you can set minimal DML logging at the database-level but a message appears, warning you that objects marked for replication still use full logging:

```
Warning: Database database_name has objects marked  
for replication. Replicated objects will continue to  
use full logging.
```

- If a database using full DML logging contains tables defined to use minimal logging, and you mark the database for replication, you see this warning:

```
Warning: Database database_name has tables that use  
minimal DML logging. These tables will not be  
replicated.
```

- If you create a table using full logging and you mark the table for replication, and then you set minimal DML logging for the table, the attempt fails and you see:

```
Cannot alter the table table_name to set minimal DML  
logging because this table is marked for  
replication. Remove replication and try again.
```

- If you create a table using minimal DML logging, and if you mark the table for replication, the attempt fails and you see:

```
Cannot set replication for table table_name because  
it is using minimal logging. Use ALTER TABLE to set  
full logging and try again.
```

Replication Server Unsupported Operations

There are some limitations when using Replication Server unsupported operations.

These Adaptive Server operations may cause incorrect replication:

- Replication Server does not support nested transactions within replicated stored procedures.

When you enable replication for a stored procedure using **sp_setrepproc** or **sp_setreplicate**, Adaptive Server always runs the stored procedure within a transaction. If you have not explicitly run the replicated stored procedure within a transaction, Adaptive Server places an implicit **begin transaction** command at the start of the procedure.

If the replicated stored procedure contains nested transaction commands such as **begin transaction**, **commit transaction**, or **rollback transaction**, you might get errors when you run the procedure. For example, a **rollback transaction** command rolls back to the start of

the stored procedure, rather than to the nested **begin transaction** command, which was the intended rollback point.

- Data that is inserted into a primary table using an unlogged bulk copy operation is not replicated.
- To use the atomic method of subscription materialization:
 - The user who enters the **create subscription** command or the database owner must own the primary table. Alternatively, you must use user-defined function strings for **select** operations at the primary database.
 - If the database owner or maintenance user does not own the replicate table, 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, create a unique function string by using a distinct function-string class. See *Customize Database Operations* in the *Replication Server Administration Guide Volume 2*.

See also

- *Replicated Function Restrictions* on page 380
- *Manage Subscriptions* on page 397

Manage Routes

A route is a one-way message stream from a source Replication Server to a destination Replication Server.

From each source Replication Server, you create one route for each destination Replication Server, no matter how many databases are managed by the source or destination Replication Servers.

Routes carry:

- Data modification commands and applied functions or applied stored procedures from primary databases managed at the source Replication Server to replicate databases managed at the destination Replication Server
- System table modification commands from a source Replication Server RSSD to a destination Replication Server RSSD
- Request functions or request stored procedures from replicate databases to primary databases (in this case, the source is the replicate Replication Server and the destination is the primary Replication Server).

When you create a route, the source Replication Server:

- Creates an RSI outbound queue to hold messages for the destination site
- Starts an RSI thread that logs in to the destination Replication Server and transfers transactions from the RSI outbound queue to the destination Replication Server

Routing Preparation

Before you create or modify routes, you must verify that the relevant components of your replication system are running, and you have designed your routes.

Specifically, you need to:

- Ensure that you have carefully determined where routes are needed in your system. As part of the design process, you must know where each source Replication Server and its destination Replication Servers reside.
- Identify which routes are direct and which are indirect. Indirect routes carry messages to destination Replication Servers through one or more intermediate Replication Servers. Using direct versus indirect routes can have a noticeable effect on system performance.
- If you are creating a direct route, define the destination Replication Server in the interfaces file at the site of the source Replication Server.
Verify you have an interfaces file entry for the RSSD of the destination Replication Server.

Manage Routes

- Ensure the source, destination, and any intermediate Replication Servers in the route are running.
- Ensure that the RepAgent thread for the source Replication Server RSSD is running.

See the *Replication Server Design Guide*.

See also

- *Routing Schemes* on page 167

Routing Rules

After you have determined your routing scheme, you can set up the required routes based on the routing rules.

The routing rules are:

- Replication Servers that manage databases containing primary data require direct or indirect routes to the Replication Servers that manage databases with subscriptions for the data.
- Replication Servers that manage replicate databases where request functions originate require direct or indirect routes to the Replication Server managing the primary database. If no replicated functions originate in the replicate database, a route from a replicate to a primary Replication Server is not required.
- Each route in an indirect route must be a direct route.
- You customize function strings for system functions with class scope at the primary Replication Server for the function-string class. In this instance, you must create routes from the primary Replication Server to the Replication Server managing the databases that use the function strings.

See Replication Server Administration Guide Volume 2 > Customize Database Operations > Work with Functions, Function Strings, and Classes > Summary of System Functions > System Functions with Function-string-class Scope.

- You customize error classes at the primary Replication Server. In this instance, you must create routes from the primary Replication Server to the Replication Server managing the databases that use the error mappings.
- A Replication Server that you plan to assign as the new primary site for a function-string class or error class, using the **move primary** command, has the following requirements:
 - It must have routes to and from the Replication Server that is the current primary site for the class, and
 - It must have routes to all the same Replication Servers as the Replication Server that is the current primary site for the class

See Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions > Data Server Error Handling > Changing the Primary Replication Server for an Error Class and Replication Server Administration Guide Volume 2 > Customize Database Operations

> *Manage Function-string Classes* > *Create a Function-string Class* > *Primary Site for a Function-string Class.*

Routing Schemes

Replication Server supports direct, indirect, and dedicated routes.

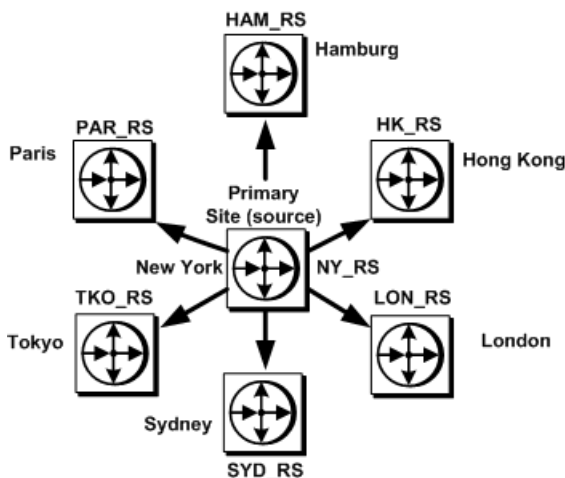
Direct Routes

A route with no intermediate sites is called a direct route. A system with direct routes results in network connections between source and destination Replication Servers.

For example, in this figure, a seven-site enterprise is shown in a star configuration, with one primary site and six replicate sites. Each replicate site has a route originating at the primary site. All six routes from the primary site are direct. Thus, the primary Replication Server has six stable queues and six RSI threads connected through the network to the six replicate sites.

If the replicate site TKO_RS is to submit request functions to the primary site NY_RS, your system would also require a direct route from TKO_RS to NY_RS, in addition to the direct route from NY_RS to TKO_RS.

Figure 11: Sites Connected with Direct Route Configuration



Indirect Routes

A route with intermediate sites is called an indirect route.

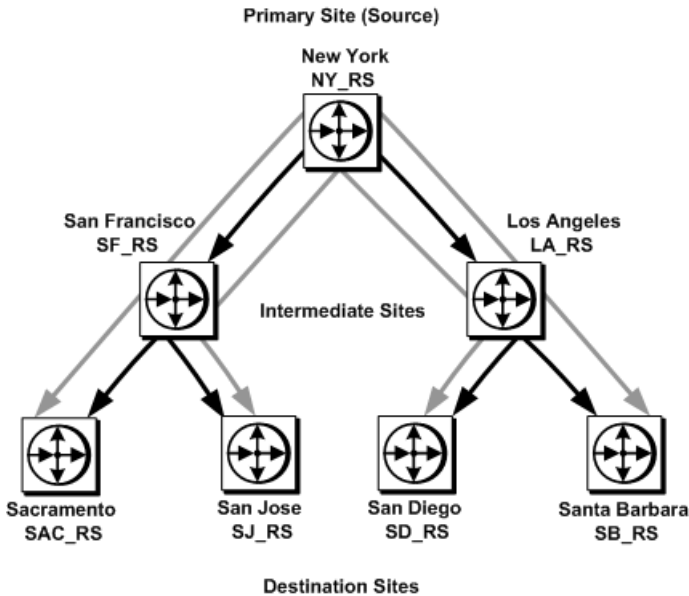
The example for indirect routes shows a seven-site enterprise with a single primary site and six replicate sites. Each replicate site has a route originating at the primary site. Only two routes

Manage Routes

from the primary site are direct; four are indirect. The two intermediate sites each have two direct routes.

In this example, NY_RS to SAC_RS is an indirect route, based on the direct routes NY_RS to SF_RS and SF_RS to SAC_RS. In an indirect route, the source Replication Server sends messages for the destination Replication Server to an intermediate Replication Server, which makes use of a route (direct or indirect) to the destination Replication Server.

Figure 12: Sites Connected with Indirect Routes in a Hierarchical Configuration



To create an indirect route, you create direct routes between each successive Replication Server along the intended indirect route. Once all the direct routes are in place, then you create the indirect route itself.

For example, to create the indirect route NY_RS to SAC_RS, first create the direct routes NY_RS to SF_RS and SF_RS to SAC_RS. Then create the indirect route based on the existing direct routes.

By setting up indirect routes, you reduce the amount of processing at the primary site and distribute the load among intermediate Replication Servers.

Table 10. Direct and Indirect Routes Between Sites in the Example

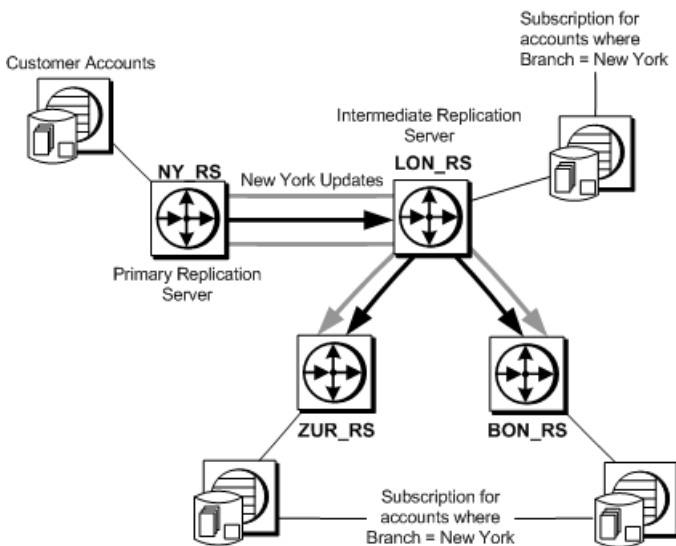
Direct Routes	Indirect Routes
NY_RS to SF_RS	NY_RS to SAC_RS
NY_RS to LA_RS	NY_RS to SJ_RS

Direct Routes	Indirect Routes
SF_RS to SAC_RS	NY_RS to SD_RS
SF_RS to SJ_RS	NY_RS to SB_RS
LA_RS to SD_RS	
LA_RS to SB_RS	

When you use indirect routes, the primary Replication Server can route portions of subscriptions that are common to destination sites through the same intermediate site. When subscriptions overlap, the primary Replication Server is required to send only one message per row modification to the intermediate Replication Server that is common to the destination sites.

In this example of sites with overlapping subscriptions, the intermediate Replication Server in LON_RS receives row modification changes for customer accounts whenever changes occur at the bank headquarters in New York. The New York modifications are also required at branch bank replicate sites in Zurich and Bonn. Because LON_RS is set up to distribute changes to ZUR_RS and BON_RS, the NY_RS primary Replication Server sends only one copy of each change to LON_RS. The number of direct routes is also reduced through the use of the two indirect routes, NY_RS to ZUR_RS and NY_RS to BON_RS.

Figure 13: Sites with Overlapping Subscriptions



Although indirect routes are helpful for distributing computing resources among sites on the network, overall propagation of data is slowed somewhat because messages are queued by more than one Replication Server. It is better to use direct routes when there are few replicate

Manage Routes

sites. When using indirect routes, minimize the number of intermediate sites to obtain the best propagation times.

Dedicated Routes

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

Note: You can only create a dedicated route between two Replication Servers if there is a direct route between the two Replication Servers. You cannot create a dedicated route if there is only an indirect route between the Replication Servers.

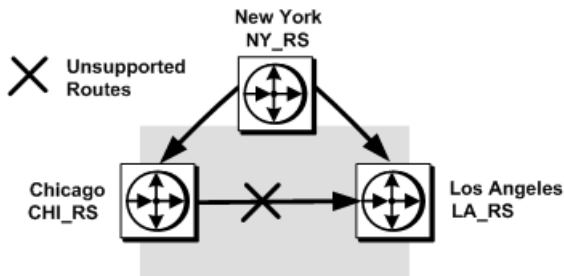
See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Dedicated Routes*.

Unsupported Routing Schemes

An intermediate Replication Server can accept transactions from one or more Replication Servers. Replication Server, however, does not support routing schemes in which routes diverge from the same source Replication Server, then converge at the same intermediate or destination Replication Server..

In this example, only one route from NY_RS to LA_RS can be supported. If the route from NY_RS to LA_RS is supported, then the route between CHI_RS to LA_RS is not supported.

Figure 14: Example of Supported and Unsupported Routes



Create Routes

Create routes at the source Replication Server.

As soon as you create a direct route between a source and destination Replication Server, the source Replication Server:

- Creates an RSI outbound stable queue to hold messages for the destination site, and
- Starts an RSI thread that logs in to the destination or next Replication Server in the route.

Note: You can create a route from a version 15.0 Replication Server to an older Replication Server (version 11.03 or later).

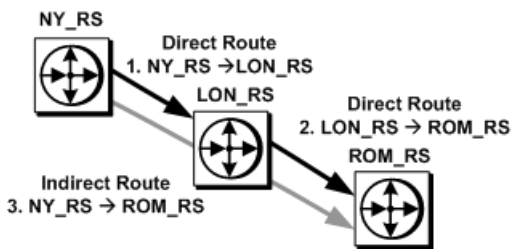
When you create either direct or indirect routes, the destination Replication Server creates and materializes subscriptions at the destination site for the replicated RSSD system tables. This process lets the destination Replication Server receive available replication definitions and function classes.

When you create an indirect route, Replication Server does not create an RSI queue. The indirect route uses the RSI outbound queues of the direct route segments that compose the indirect route.

For Replication Server to be able to begin transferring system information to the destination Replication Server, you must create direct routes before you create an indirect route.

For example, you cannot create an indirect route (1 to 3) unless you have already created two direct routes (1 to 2 and 2 to 3). You also must set up the routes in the correct order.

Figure 15: Order for Creating Direct and Indirect Routes



See also

- *Replication Server Technical Overview* on page 23

create route Command

There are several prerequisites, guidelines, and configuration parameters for **create route**.

Note: You can also create routes in Sybase Central.

The syntax for the **create route** command is:

```
create route to dest_replication_server{
  with primary at dataserver.database|
  set next site [to] thru_replication_server|
  [set username [to] user]
  [set password [to] passwd]
  [set route_param to 'value']
  [set security_param to 'value']}
```

When creating routes:

Manage Routes

- Supply the login name, password, and other parameters for direct routes only.
- Before you create a direct route, create its login name and password in the destination Replication Server. Optionally, you can have the **rs_init** utility create this user.
 - If you are enabling network-based security and unified login, user name and password are optional. The default user name is the principal user name, which is specified by the **-S** flag when you log in to Replication Server or start Replication Server.
 - If you create a route with a *user* and *passwd* that do not exist at the destination Replication Server, add or change the *user* and *password* at that destination.
 - If you are establishing a direct route from the current Replication Server to the destination Replication Server, do not use the **next site** clause.
- Use the **with primary at** clause to create a dedicated route.

Note: You can only create a dedicated route between two Replication Servers if there is a direct route between the two Replication Servers. You cannot create a dedicated route if there is only an indirect route between the Replication Servers.

- Enter one **create route** command at a time, to ensure you have made no mistakes. Wait for a route to become valid before creating the next one.

If you do make a mistake, drop the route and re-create it only as a last resort. Include the **with nowait** option with the **drop route** command. Since the route has not been created, its current state requires that you use the **with nowait** option to drop it.

When you create a route, you can accept the default values for configuration parameters that manage memory size, the size of the amount of data that can be sent over the route at one time, time-outs, and synchronization intervals. You can also set your own values when you create or alter the route.

If network-based security is enabled at your site, you can also configure security parameters for routes. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance* for a list and discussion of route parameters that affect performance.

See also

- *Establish the Principal User* on page 277
- *Drop Routes* on page 184
- *Manage Network-based Security* on page 261
- *Changing an Indirect Route to a Direct Route* on page 179

Configuration Parameters Affecting Routes

There are several parameters you can use to configure routes.

Table 11. Configuration Parameters Affecting Routes

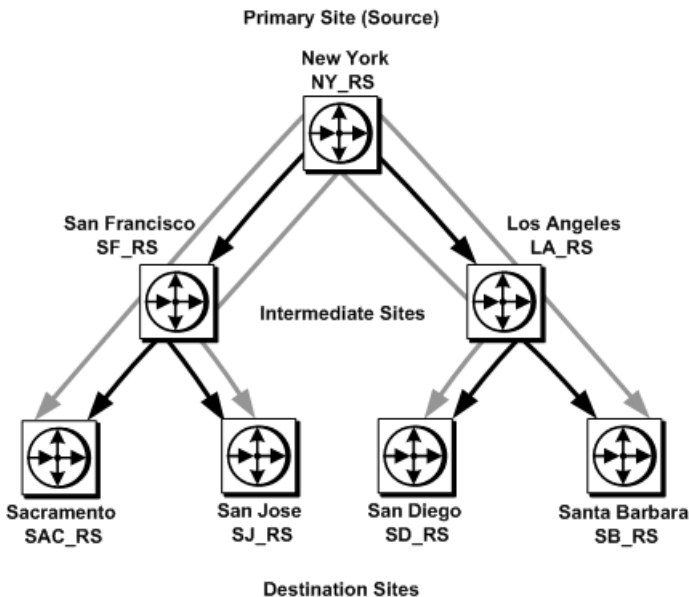
route_param	value
disk_affinity	Specifies an allocation hint for assigning the next partition. Enter the logical name of the partition to which the next segment should be allocated when the current partition is full. Values are " <i>partition_name</i> " and "off." Default: off
rsi_batch_size	The number of bytes sent to another Replication Server before a truncation point is requested. Default: 256K Minimum: 1K 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 (Replication Server does not close the connection)
rsi_packet_size	Packet size, in bytes, for communications with other Replication Servers. The range is 1024 to 16384. Default: 4096 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. The value 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. Default: 0 minutes

Examples of Creating Direct and Indirect Routes

Use the examples to learn to create direct and indirect routes, and set configuration parameters.

You need to create the direct routes from the primary Replication Server to the intermediate Replication Server and from the intermediate Replication Server to the destination Replication Server before you can create an indirect route.

Figure 16: Sites Connected with Indirect Routes in a Hierarchical Configuration



Example 1

To create the direct route from NY_RS, the primary Replication Server, to SF_RS, enter in NY_RS:

```
create route to SF_RS
set username SF_rsi_user
set password SF_rsi_ps
```

Example 2

To create the direct routes SF_RS to SAC_RS and SF_RS to SJ_RS, enter in the intermediate Replication Server, SF_RS:

```
create route to SAC_RS
set username SAC_rsi_user
set password SAC_rsi_ps
```



```
create route to SJ_RS
set username SJ_rsi_user
set password SJ_rsi_ps
```

Example 3

After the direct routes are created, you can create indirect routes through them.

The following example creates the indirect routes from the primary site NY_RS to sites SAC_RS and SJ_RS, through the intermediate site, SF_RS. Enter these commands in the primary Replication Server, NY_RS:

```
create route to SAC_RS
set next site SF_RS
```

```
create route to SJ_RS
set next site SF_RS
```

Example 4

You can create a route and configure parameters at the same time.

To set the `rsi_packet_size` to 4096 bytes for the route to SF_RS, enter:

```
create route to SF_RS
set username SF_rsi_user
set password SF_rsi_ps
set rsi_packet_size to '4096'
```

See also

- *Indirect Routes* on page 167

Configuring a Replication Server to Manage Primary Tables

You can add a route from a Replication Server that was previously configured as a replicate-only Replication Server.

You must first set up the RepAgent for the Replication Server RSSD. Any database that functions as a primary database also requires a RepAgent. You must configure RepAgent at the Replication Server and at the primary Adaptive Server database.

1. Configure RepAgent at the Replication Server:

- a) Create a RepAgent user so that RepAgent can log in to Replication Server. Use the **create user** command where `ra_user_name` is the name of the RepAgent user and `ra_password` is the RepAgent password:

```
create user ra_user_name
set password {ra_password | null}
```

- b) Grant this user **connect source** permission, using the **grant** command:

```
grant connect source to ra_user_name
```

Manage Routes

If the Replication Server already manages a primary database, you can use the “RepAgent user” that already exists for the new primary database.

- c) Execute **alter connection** using the **log transfer on** option:

```
alter connection to data_server.database
set log transfer to 'on'
```

2. Configure RepAgent at the Adaptive Server database:

- a) If the name of the Adaptive Server has not yet been defined, you must define it using the following command where *lname* is the RSSD name:

```
sp_addserver lname, local
```

- b) If RepAgent threads have not been enabled for the Adaptive Server, you must enable them:

```
sp_configure 'enable rep agent threads'
```

- c) Configure RepAgent for the RSSD with the **sp_config_rep_agent** system procedure:

```
sp_config_rep_agent dbname, 'enable', 'rs_name',
                    'rs_user_name', 'rs_password'
```

Note: The value for *rs_user_name* and *rs_password* configured at the Adaptive Server must be the same for *ra_user_name* and *ra_password* created at the Replication Server in step 1.

- d) Start RepAgent:

```
sp_start_rep_agent dbname
```

See also

- *Configuring RepAgent* on page 113

Suspend and Resume Routes

When you alter a direct route, change its topology, or perform some other maintenance to a remote site, you must suspend the route so that messages are no longer sent to the destination Replication Server. After maintenance is completed for the route, you can then reactivate the route to resume activity.

You can suspend and resume routes in Sybase Central or with the **suspend route** and **resume route** RCL commands.

You can suspend and resume dedicated routes with **suspend route** and **resume route**. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Dedicated Routes*.

suspend route

suspend route suspends a route to another Replication Server.

While a route is suspended, no messages are sent to the destination Replication Server, and the messages for the Replication Server are held in a stable queue. The syntax for the **suspend route** command is:

```
suspend route to dest_replication_server
[with primary at dataserver.database]
```

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

For example, to suspend the route to the CHI_RS Replication Server, enter:

```
suspend route to CHI_RS
```

resume route

resume route resumes a suspended route.

Resuming a route allows the source Replication Server to begin sending queued messages to the destination Replication Server. You can also use this command to resume a route that was suspended automatically as the result of an error. The syntax for the **resume route** command is:

```
resume route to dest_replication_server
[with primary at dataserver.database |
skip transaction with large message]
```

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

For example, to resume the route to the CHI_RS Replication Server, enter:

```
resume route to CHI_RS
```

Change Routes

Use **alter route**, or Sybase Central to change the topology, user name, password, and certain configuration parameters of a direct route.

Note: You cannot change the parameters of an indirect route with **alter route**.

The syntax for **alter route** is:

```
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'}
```

To alter a route, you must:

1. Suspend the route.

Manage Routes

2. Execute **alter route** with the relevant configuration parameters.
3. Resume the route for the changes to take effect.

See also

- *Manage Network-based Security* on page 261

Change Route Topolgy

You can modify the route topology by changing a direct route to an indirect route, changing an indirect route to a direct route, or changing the next intermediate site for an indirect route.

Changing a Direct Route to an Indirect Route

Change an existing direct route to an indirect route.

1. At the source Replication Server, from which the direct route originates, enter:

```
suspend route to dest_replication_server
```

2. At each Replication Server that manages a database with a RepAgent, enter:

```
suspend log transfer from all
```

and then quiesce the replication system so that messages will be redirected to your new routing configuration without error.

3. Create any additional routes that the new indirect route will use.
 - If the current Replication Server does not already have a direct route to the Replication Server that you will specify as the intermediate site for the new indirect route, create the route.
 - If the Replication Server that you will specify as the intermediate site for the new indirect route does not already have a direct or indirect route to the destination site, create the route.
4. For the direct route you are changing to an indirect route, enter the following command at the source Replication Server where *dest_replication_server* is the destination Replication Server for the route you are altering, and *thru_replication_server* is the intermediate Replication Server for the route:

```
alter route to dest_replication_server  
set next site [to] thru_replication_server
```

5. Resume log transfer connections by entering the following command at each Replication Server where you previously suspended log transfer:

```
resume log transfer from all
```

6. At the source Replication Server, resume the suspended route by entering the following command:

```
resume route to dest_replication_server
```

See also

- *Quiescing a Replication System* on page 105

Changing the Next Intermediate Site for an Indirect Route

Change the next intermediate site for an existing indirect route.

1. Enter the following command at the source Replication Server, from which the direct route originates:

```
suspend route to dest_replication_server
```

2. At each Replication Server that manages a database with a RepAgent, enter:

```
suspend log transfer from all
```

and then quiesce the replication system so that messages will be redirected to your new routing configuration without error.

3. Create any additional routes that the indirect route will use.
 - If the current Replication Server does not already have a direct route to the Replication Server that you will specify as the new intermediate site for the indirect route, create the route.
 - If the Replication Server that you will specify as the new intermediate site for the indirect route does not already have a direct or indirect route to the destination site, create the route.

4. For the indirect route for which you are specifying a new intermediate Replication Server, enter the following command at the source Replication Server where *dest_replication_server* is the destination Replication Server for the route you are altering, and *thru_replication_server* is the new intermediate Replication Server for the route:

```
alter route to dest_replication_server  
set next site thru_replication_server
```

5. Resume log transfer connections by entering the following command at each Replication Server where you previously suspended log transfer:

```
resume log transfer from all
```

6. Resume the suspended route by entering the following command at the source Replication Server:

```
resume route to dest_replication_server
```

See also

- *Quiescing a Replication System* on page 105

Changing an Indirect Route to a Direct Route

Change an existing indirect route to a direct route.

1. Enter the following command at the source Replication Server, from which the indirect route originates:

```
suspend route to dest_replication_server
```

2. At each Replication Server that manages a database with a RepAgent, enter:

```
suspend log transfer from all
```

Manage Routes

and then quiesce the replication system so that messages will be redirected to your new routing configuration without error.

3. For the indirect route you are changing to a direct route, enter the following command at the source Replication Server where *dest_replication_server* is the destination Replication Server for the route you are altering, and *user* and *passwd* are the RSI user login name and password to use for the direct route:

```
alter route to dest_replication_server
set username user set password passwd
```

4. Resume log transfer connections by entering the following command at each Replication Server where you previously suspended log transfer:

```
resume log transfer from all
```

5. Resume the suspended route by entering the following command at the source Replication Server:

```
resume route to dest_replication_server
```

See also

- *Quiescing a Replication System* on page 105

Changing the Password for the RSI User for a Direct Route

Use **alter route** with **set password** to change the RSI user password for an existing direct route.

1. Suspend each direct route from the source Replication Server by entering:

```
suspend route to dest_replication_server
```

2. At the source Replication Server, enter the following command where *dest_replication_server* is the destination Replication Server for the route you are altering, and *passwd* is the password to use for the RSI user login name:

```
alter route to dest_replication_server
set password passwd
```

3. Resume each suspended route from the source by entering:

```
resume route to dest_replication_server
```

Changing Parameters Affecting Direct Routes

Use **alter route** or Sybase Central to change parameters for a specific direct route after you create the route.

Configuration parameters set for individual routes with **alter route** override default parameters set with **configure replication server**. Thus, you can set default parameters with **configure replication server** and then customize settings for individual routes with **alter route**.

For example, to change the **rsi_sync_interval** parameter to 120 seconds using **alter route**, log in to the source Replication Server and:

1. Suspend the route:

```
suspend route to dest_replication_server
```

2. Execute the **alter route** command:

```
alter route to dest_replication_server
set rsi_sync_interval to '120'
```

3. Resume the suspended route:

```
resume route to dest_replication_server
```

Configuration changes take effect after you resume the route.

See also

- *Configuration Parameters Affecting Routes* on page 173
- *Changing Configuration Parameters for All Routes* on page 181

Changing Configuration Parameters for All Routes

Use the **configure replication server** command to set default configuration parameters for all routes originating at the source Replication Server.

Configuration parameters set for individual routes with **alter route** override default parameters set with **configure replication server**. Thus, you can set default parameters with **configure replication server** and then customize settings for individual routes with **alter route**.

The syntax for changing route parameters with **configure replication server** is:

```
configure replication server
set route_param to 'value'
```

Here is an example of using **configure replication server** to change the **rsi_save_interval** parameter to 2 minutes. To execute the command, log in to the source Replication Server and perform these steps:

1. Suspend all routes from the source Replication Server. For each route, enter:

```
suspend route to dest_replication_server
```

2. Execute the **configure replication server** command:

```
configure replication server
set rsi_save_interval to '2'
```

3. Resume suspended routes from the source Replication Server. For each route, enter:

```
resume route to dest_replication_server
```

Configuration changes take effect after you resume the routes.

See also

- *Configuration Parameters Affecting Routes* on page 173

Routing Modification Example

Learn how to change a routing scheme.

This example shows how to change the routing scheme from the scheme in the "Sites Connected with Indirect Routes in a Hierarchical Configuration" diagram to the scheme in the "Indirect Routes Altered" diagram, where LA_RS becomes an intermediate site between NY_RS and SF_RS, while direct and indirect routes to SB_RS are dropped:

Figure 17: Sites Connected with Indirect Routes in a Hierarchical Configuration

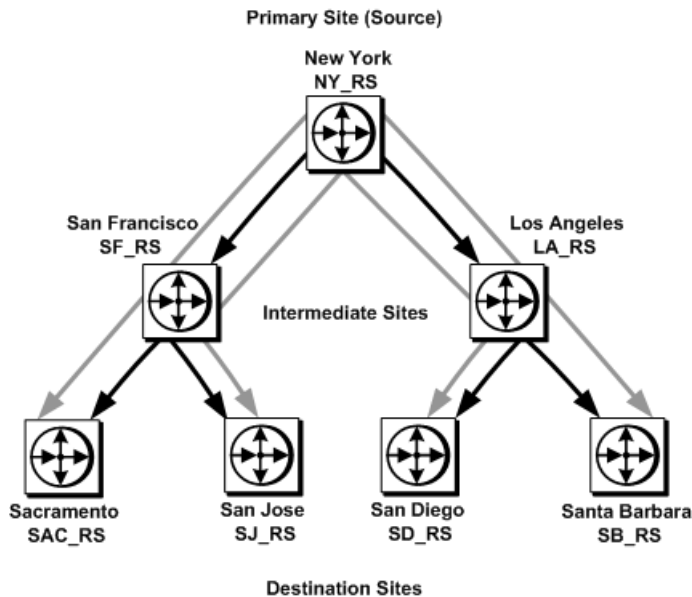
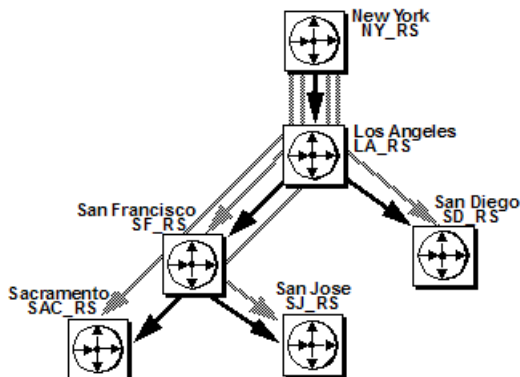


Figure 18: Indirect Routes Altered



1. At each Replication Server that manages a database with a RepAgent, enter:

```
suspend log transfer from all
```

and then quiesce the replication system so that messages will be redirected to your new routing configuration without error.

2. LA_RS needs a direct route to SF_RS; create one by entering the following command at Replication Server LA_RS:

```
create route to SF_RS
set username SF_rsi_user
set password SF_rsi_ps
```

3. LA_RS requires indirect routes to SAC_RS and SJ_RS, through SF_RS.

Creating these routes instructs LA_RS to send messages to SF_RS that are destined for SAC_RS and SJ_RS. SF_RS already has direct routes to SAC_RS and SJ_RS. Enter the commands in Replication Server LA_RS:

```
create route to SAC_RS
set next site SF_RS
```

```
create route to SJ_RS
set next site SF_RS
```

4. The primary Replication Server, NY_RS, was previously configured with indirect routes through SF_RS to SAC_RS and SJ_RS. Alter those routes so that Replication Server LA_RS is the next Replication Server. Enter these commands in Replication Server NY_RS:

```
alter route to SAC_RS
set next site LA_RS
```

```
alter route to SJ_RS
set next site LA_RS
```

5. The direct route from the primary Replication Server, NY_RS, to SF_RS needs to be changed to an indirect route, with LA_RS as the intermediate Replication Server. Enter these commands in Replication Server NY_RS:

```
alter route to SF_RS
set next site LA_RS
```

6. At each Replication Server where you previously suspended log transfer, resume log transfer connections to each Replication Server by entering:

```
resume log transfer from all
```

7. Remove the indirect route from NY_RS to SB_RS. Enter this command in NY_RS:

```
drop route to SB_RS
```

8. Remove the direct route from LA_RS to SB_RS. Enter this command in LA_RS:

```
drop route to SB_RS
```

The indirect route from NY_RS to SD_RS, through LA_RS, is intact.

See also

- *Resume Log Transfer* on page 118

- *Quiescing a Replication System* on page 105

Drop Routes

Dropping a route closes the route from the Replication Server where you execute the command to a specified remote Replication Server.

Dropping a route performs the following actions on participating Replication Servers:

- Drops system table subscriptions.
- If the route is direct, the outbound stable queue is dropped and the RSI thread is stopped.
- Deletes information regarding the route.

You cannot drop the route if:

- It is a direct route used by any indirect routes to additional destination Replication Servers.
- The source Replication Server has replication definitions that are subscribed to by the destination Replication Server.
- The source Replication Server is designated as the primary site of a function-string class or error class. The primary site of a derived function-string class is the same as its parent class.

You can monitor the status of the route while it is being dropped by one of:

- In Sybase Central, view status information in the right pane of the Sybase Central main window.
- From the command line, execute the **rs_helproute** stored procedure.

drop route Command

Use the **drop route** command to drop routes.

You can also drop routes from Sybase Central. The syntax for the **drop route** command is:

```
drop route to dest_replication_server
[with primary at dataserver.database]
[with nowait]
```

Include the **with primary at *dataserver.database*** clause in the command to specify a dedicated route, where *dataserver.database* is the primary connection name. You must drop the dedicated route before you drop a shared route. After you drop a dedicated route, transactions from the specified primary connection to the destination Replication Server go through the shared route. See *Replication Server Administration Guide Volume 2 > Performanc Tuning > Multi-Path Replication > Dedicated Routes*.

The **with nowait** option instructs Replication Server to close the route even if it is unable to communicate with the destination Replication Server.

Warning! Use the **with nowait** clause only as a last resort. Use the **with nowait** clause only if you do not intend to ever use the destination Replication Server, or if you must drop the route

from the source Replication Server while the destination Replication Server is unavailable, or if you are attempting to add or change login names and passwords for direct routes. Avoid using the **with nowait** clause whenever possible.

The clause forces Replication Server to drop a route even if the route contains transactions in the outbound queue of the route. As a result, Replication Server may discard some transactions from the primary connections. The clause instructs Replication Server to drop the dedicated route even if the route cannot communicate with the destination Replication Server.

After you use the **with nowait** clause, use the **sysadmin purge_route_at_replicate** command to remove all references to a primary Replication Server such as subscriptions and route information, from the system tables at the replicate Replication Server.

sysadmin purge_route_at_replicate Command

sysadmin purge_route_at_replicate removes all subscriptions and route information originating from a specified primary Replication Server after a route is dropped from that Replication Server.

Before you execute this command, drop the route from the replicate Replication Server to the primary Replication Server, if it exists. The Replication Server performs a validation check before it processes the command.

Execute **sysadmin purge_route_at_replicate** at the replicate Replication Server, using the following syntax where *replication_server* is the primary Replication Server:

```
sysadmin purge_route_at_replicate, replication_server
```

Upgrade Routes

Upgrading the route allows the Replication Servers to exchange information about newer software features.

The route version is the earliest site version of the source and destination Replication Server. After you upgrade the source and destination Replication Servers on either end of a route and also set their site versions to a higher Replication Server version, you need to upgrade the route.

Upgrading a route rematerializes data in system tables, making information associated with new features available to a newly upgraded Replication Server. After upgrading, new types of information that were not previously allowed can be exchanged.

To display the current version number of routes that originate or terminate at a Replication Server, use the **admin show_route_versions** command.

See *Replication Server Reference Manual > Replication Server Commands > admin show_route_versions* for complete syntax and usage information.

There are two possible scenarios for route upgrade:

Manage Routes

- If new features have not been used at the source Replication Server, use **sysadmin fast_route_upgrade** to upgrade routes.
- In all other cases, use the commands **route_upgrade**, **route_upgrade_recovery**, and **route_upgrade_status** to upgrade routes.

You cannot downgrade a route after you have upgraded it, and there are restrictions if you have mixed-version replication systems.

See the installation and configuration guides for your platform for more information about upgrading routes and setting the site version for a Replication Server.

See also

- *Mixed-Version Replication Systems* on page 17

Monitor Routes

Routes may display different statuses at different times. Acquire an overview of the utilities you can use to monitor the status of routes.

When you create a route, the destination Replication Server subscribes to the source Replication Server system tables. Depending on the volume of your data, it may take several minutes for subscriptions to materialize. Dropping a route also may take some time.

- You can use the **admin who** command to display thread status information.
- For comprehensive status information, including the current state of routes you are creating, use **rs_helproute**.

Display RSI Thread Status Using **admin who**

Use **admin who** to view RSI thread information.

- **admin who** displays all threads in the system, including RSI threads.
- **admin who, rsi** displays the status of the RSI thread, which Replication Server starts to submit information to other Replication Servers.

See *Replication Server Reference Manual > Replication Server Commands > admin who* for detailed thread status information.

Use **rs_helproute** Stored Procedure

Use **rs_helproute** to obtain comprehensive route status information, including the current state of routes you are creating.

Execute the **rs_helproute** stored procedure in the RSSD at the source or destination Replication Server for the route. The syntax for the **rs_helproute** stored procedure is:

```
rs_helproute [replication_server]
```

If you specify the name of a Replication Server, **rs_helproute** returns information only for routes for which the named Replication Server is a source or destination. Otherwise, it returns information for all routes for which the current Replication Server is a source or destination.

rs_helproute returns two types of information:

- Route status, which reflects the state of the route at the site where **rs_helproute** is executed. A route is valid when **rs_helproute** at both source and destination returns “Active.”

Other route status values are:

- Being created
- Being dropped
- Being dropped **with nowait**
- List of system table subscriptions, which tells you the system table subscriptions that are being created. If a route is being dropped, it tells you which subscriptions are being dropped.

If no system table subscriptions are listed, the route has been created and is in working order.

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

Refer to the *Replication Server Troubleshooting Guide* for information about correcting route creation problems.

Manage Database Connections

Managing database connections includes preparing databases for replication, creating connections to the databases, and configuring the connections for optimal replication performance.

Prepare Databases for Replication

Before you can add databases to a replication system, you need to prepare them so that Replication Server can distribute the primary data and maintain the replicated data stored in them.

- If your databases are managed by Sybase Adaptive Servers:
Use Sybase Central or **rs_init** to prepare Adaptive Server databases for use with Replication Server.
See the *Replication Server Installation Guide* and *Replication Server Configuration Guide* for more information on **rs_init**.
- If your databases are managed by non-ASE data servers:
Refer to the *Replication Server Design Guide* for required preparations. In addition, to find out how to prepare your database for the heterogeneous datatype support (HDS) feature, see the *Replication Server Configuration Guide* for your platform. HDS enables the translation of primary database column values of one datatype to another datatype acceptable to the replicate database.

When you are connecting a new database to an existing system, always conduct a careful review and analysis of how the database will fit into your system. Determine which other processes are required for the database, and designate required names and login names for these processes.

If you anticipate that an existing “replicate-only” database may in the future be the source of replicated function delivery or contain primary data, you can set up the database so that it can manage primary tables. You can then avoid upgrading the replicate-only database in the future.

See also

- *Translate Datatypes Using HDS* on page 372

Preparation of Adaptive Server Databases for Replication

To prepare Adaptive Server databases for replication, use Sybase Central or **rs_init**.

Sybase Central or **rs_init**:

Manage Database Connections

1. Creates the `rs_lastcommit` system table.
2. Loads the `rs_update_lastcommit` and `rs_get_lastcommit` stored procedures (for both primary and replicate databases) and the `rs_marker` stored procedure (for primary databases only).
3. Creates the `rs_threads` system table.
4. Loads the `rs_initialize_threads` and `rs_update_threads` stored procedures for the database.
5. Creates the maintenance user login name and verify that the maintenance user can log in to the database.
6. Creates a connection from Replication Server to the database, allowing Replication Server to manage the database.

If the database has primary data, Sybase Central or `rs_init` additionally:

- Enables RepAgent at the Adaptive Server.
- Enables and configures RepAgent at the database.
- Sets the secondary truncation point to “valid” in the Adaptive Server database, preventing Adaptive Server from truncating database log records before RepAgent has read them.
- Creates the RepAgent user name and password in the Replication Server, if necessary.
- Starts RepAgent.

Refer to the Replication Server installation and configuration guides for your platform for details on each step.

See also

- *Manage the Maintenance User* on page 192

Preparation of Non-ASE Servers for Replication

With Replication Server 15.2, you can use connection profiles to connect to non-ASE servers with simplified installation and configuration.

To prepare non-ASE databases for replication, see the *Replication Server Heterogeneous Replication Guide*.

You do not need to edit and execute scripts to install datatype definitions, function strings, and class-level translations for heterogeneous (non-ASE) datatype support. The functions provided by the scripts are included as part of the Replication Server installation, or are included in connection profiles that are installed with Replication Server. These enhancements simplify installation and configuration for heterogeneous and non-ASE environments. See *Replication Server Configuration Guide > Install and Implement Non-ASE Support Features*.

In addition, Replication Server 15.2 and later provides error class support for non-ASE replicate databases. See *Replication Server Administration Guide Volume 2 > Handle Errors and Exceptions > Data Server Error Handling*.

Connection Profiles

Connection profiles contain connection configurations and replicate database object definitions relevant to each type of actively supported non-ASE data server.

You can use these connection profiles together with simple syntax to create connections between actively supported data servers such as Adaptive Server Enterprise, HANA DB, IBM DB2, Microsoft SQL Server, and Oracle. Replication Server uses the connection profile to configure the connection and create replicate database objects for you.

Connection profiles specify the function-string class, error class, and class-level translations to be installed. You can also use connection profile options to specify other actions such as whether commands should be batched and the command separator to use.

Note: When you create a connection using a connection profile, the system table services (STS) caches are refreshed so that you do not need to restart Replication Server.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Command Batching for non-ASE Servers*.

See also

- *Create Database Connections* on page 194

Upgrade an Existing Adaptive Server Database

You may need to upgrade a database to work with the latest version of Replication Server so that you can use newer features. Use **rs_init** to upgrade a database.

Upgrading a database ensures that the database maintenance user has the Replication role and the necessary permissions (**update**, **insert**, and **delete**) in the database. The Replication role gives the maintenance user authorization to execute any necessary replication-related Adaptive Server commands.

You can check the authorizations that have been granted to a database by using the **sp_displaylogin** system procedure in the database.

Granting Replication Role to the Maintenance User

Use **sp_role** to grant replication role to the maintenance user.

In the database, execute:

```
sp_role "grant", replication_role, maintenance_user
```

Granting Permissions on Tables

Use **grant all** to grant permissions on tables in the database

For each table, execute in the database:

```
grant all on table_name to maintenance_user
```

Manage the Maintenance User

To update replicated data, Replication Server logs in to the data server as the maintenance user. The database owner or the system administrator must grant to the maintenance user the permissions required to insert, delete, and update rows in replicated tables and to execute replicated stored procedures.

Initially, Sybase Central or **rs_init** creates the login name for the maintenance user and adds the user to the replicate database. For details, refer to the Replication Server installation and configuration guides for your platform.

The maintenance user login name and password are provided to Replication Server with the **create connection** command for the database. Sybase Central or the **rs_init** program executes this command automatically. If you change the password for the login name in the data server, use Sybase Central or the **alter connection** command to change the password for the Replication Server connection.

You cannot use the **rs_init** utility for non-ASE databases. You must create and manage maintenance user login names in the non-ASE database server. See the *Replication Server Heterogeneous Replication Guide* for information related to non-ASE servers.

For Adaptive Server databases, you can protect the maintenance user with additional security measures.

See also

- *Maintenance User Security* on page 235

Find the Current Maintenance User

Use **rs_helpuser** to determine the login name that is currently assigned as maintenance user for a database.

Enter the **rs_helpuser** Adaptive Server stored procedure at the RSSD, where *user* is the login name about which you want information:

```
rs_helpuser [user]
```

Grant Permissions in the Database

Use Sybase Central or **rs_init** to grant the maintenance user permission to access the **rs_lastcommit** system table and the stored procedures that use it. Use **grant all** to grant permissions for tables and stored procedures involved in replication.

Neither Sybase Central nor **rs_init** grants permissions to the maintenance user for user tables and stored procedures. You must grant permissions on replicated tables and stored procedures before you can either replicate transactions for replicated tables or replicate executions of the replicated stored procedures.

For each table that is replicated in the database, and for each stored procedure that is executed due to replication, execute this **grant** command:

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

Note: Among the permissions granted to the maintenance user is **replication_role**. If you revoke this permission, you will not be able to replicate **truncate table** unless the maintenance user has been granted **sa_role**, owns the table, or is aliased as the database owner.

See the *Replication Server Heterogeneous Replication Guide* for the permissions to grant the maintenance user for non-ASE replicate databases.

Grant Permissions for a Primary Database

If a replicate database holds primary data, then it is also a primary database. In a primary database, special permissions are necessary on two replication objects: subscriptions and request functions.

When subscriptions are created, the **rs_marker** stored procedure is executed at the primary database. Any database user who can create subscriptions must have permission to execute **rs_marker**.

A primary database may also receive transactions via request function delivery from clients at replicate sites. These transactions are executed at the primary site as if by the user executing the request function. Any user login name with permission to execute request functions must also have permission to execute **rs_update_lastcommit**, which executes in every DSI transaction.

The permission requirements are the same for request functions and request stored procedures.

The following **grant** commands allow any user in the database to execute **rs_marker** and **rs_update_lastcommit**:

```
grant execute on rs_marker to public
grant execute on rs_update_lastcommit to public
```

These stored procedures should only be executed by Replication Server on behalf of users. Sybase Central or **rs_init** grants these permissions to “public.” You may want to restrict permissions to the database users who are allowed to create subscriptions, execute request functions, or request stored procedures.

See also

- *Manage Replicated Functions* on page 379

Create Database Connections

A connection defines a database to the Replication Server. A Replication Server is designated to manage the database and, if it is a replicate database, to distribute transactions to the database.

The database connection provides Replication Server with:

- The name of the data server and database the connection is for
- The error class used to process errors returned from the data server
- The function-string class to use with the database
- The maintenance user login name and password
- Information about whether there is a RepAgent thread for the database connection
- Options for creating active and standby databases for warm standby applications
- Configuration parameters that affect connections

To create:

- A standard connection to an Adaptive Server database, use Sybase Central, **rs_init**, or **create connection**.
- Alternate connections to an Adaptive Server database, use **create alternate connection**. For example, you can create alternate connections to build multiple replication paths. See:
 - *Replication Server Reference Manual > Replication Server Commands > create alternate connection*
 - *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication*
- Connection to a non-Sybase database, use **create connection** with the **using profile** clause. See *Replication Server Reference Manual > Replication Server Commands > create connection using profile*.

Information Required to Add a Database Connection

You need to specify several items when you add a database connection.

The Replication Server installation and configuration guides for your platform describe how you use **rs_init** to add databases.

When you add a database, you specify:

- Replication Server name
- Replication Server System Administrator user name and password
- Adaptive Server name
- Adaptive Server System Administrator user name and password
- Database name

- Whether the database requires a RepAgent
- Maintenance user name and password
- Database Owner user name and password
- Whether the physical connection is for an existing logical connection

You cannot use the **rs_init** utility for non-ASE databases. You must create and manage maintenance user login names in the non-ASE database server, and you can create connections to non-ASE databases using connection profiles.

See the *Replication Server Heterogeneous Replication Guide* and the Replication Server Options documentation to configure access to non-ASE databases.

See also

- *Connection Profiles* on page 191

Add Databases for Logical Connections

If you are adding a physical connection for an existing logical connection, which you create with Sybase Central or the **create logical connection** command, you must specify additional information.

Specify:

- Active or standby connection
- Logical data server name
- Logical database name

In addition, if you are adding a standby connection, in Sybase Central or **rs_init** you specify:

- Active data server name
- Active database name
- Active database System Administrator user name and password
- Whether to initialize standby database using dump and load method
- Whether to use dump marker to start replication

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications* .

Add a Database that Requires a RepAgent Thread

If you are adding an Adaptive Server primary database that requires a RepAgent, you must specify the Replication Server user name and password.

Use the create connection Command

To add a database for a non-ASE data server, use the **create connection** command with a connection profile specific to your non-ASE data server. To add a database for an Adaptive

Manage Database Connections

Server data server, you use Sybase Central or **rs_init**, both of which prepare the database for replication.

You can also use **create connection** for Adaptive Server data servers.

If you use **create connection** without a connection profile, you must prepare the database for replication yourself.

Enter **create connection** at the Replication Server that is to manage the database. The syntax is:

```
create connection to data_server.database
  using profile connection_profile; version
  set username [to] user
  [set error class [to] error_class]
  [set function string class [to] function_class]
  [set password [to] passwd]
  [set database_param [to] 'value']
  [set security_param [to] {'required' | 'not_required'}]
  [with {log transfer on, dsi_suspended}]
  [as active for logical_ds.logical_db |
  as standby for logical_ds.logical_db]
  [use dump marker]
  [display_only]
```

You must use the **with dsi_suspended** clause, which starts the connection with the DSI suspended, when you create a connection to a database that will not be a replicate database.

The **as active**, **as standby**, and **use dump marker** clauses are used only when you create physical connections for a logical connection for a warm standby database. Only Adaptive Server and Oracle databases may be used in warm standby applications.

If your system supports network-based security, use the **set security_param** command.

See also

- *Prepare Databases for Replication* on page 189
- *Manage Network-based Security* on page 261

using profile Clause

Use the **using profile** clause with **create connection** to specify one of the connection profiles installed with Replication Server for your non-ASE database.

The **using profile** clause uses predefined information in the connection profile you specify, to configure the connection between Replication Server and a non-Adaptive Server database, and, if needed, to modify the RSSD and the named *data_server.database*

Since the connection profile specifies the function string class, error class, and class-level translations, you do not need to specify the corresponding clauses in the **create connection** command for non-ASE databases. Use *version* to specify the particular version of the connection profile you want to use.

For example, to create a connection to an Oracle replicate database with the **rs_ase_to_oracle** connection profile:

```
create connection to oracle_db.Oracle_DS
using profile rs_ase_to_oracle;eco
set username to ora_maint
set password to ora_maint_pwd
```

Use the **admin show_connection_profiles** command to list the connection profile name, version, and comments for each profile defined in Replication Server, and use the *match_string* option to display only the connection profiles whose names contain the string you provide in the option:

```
admin show_connection_profiles[, "match_string"]
```

.

You can only use the **ase_to_ase** profile to create a connection to a replicate Adaptive Server database.

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

See *Replication Server Heterogeneous Replication Guide > Heterogeneous Warm Standby for Oracle > Setting Up Warm Standby Databases > Creating Connection to the Standby Database*.

Altering Database Connections

Use Sybase Central or **alter connection** to change the attributes of a database connection at the Replication Server where the connection was created.

If you use **alter connection**, you must suspend and resume the connection for the changes to take effect.

1. Use **suspend connection** to suspend activity on the connection.
2. Execute the **alter connection** command with the relevant parameters.

Note: Using the **set log transfer off** clause for the **alter connection** command drops the RepAgent connection from a primary site. Before using this clause, be sure there are no replication definitions defined for data in the database.

3. Use **resume connection** to resume activity on the connection.

See also

- *Suspend Database Connections* on page 198
- *Set and Change Parameters Affecting Physical Connections* on page 198
- *Resume Database Connections* on page 224

Suspend Database Connections

You must suspend a database connection before you alter it or when you remove a data server from service for maintenance.

If you have **sa** permission, you can temporarily suspend access to a data server.

While data server access is suspended, the Replication Server queues transactions for the data server so they can be applied when the connection is resumed.

You can temporarily suspend access to a data server using Sybase Central or you can enter:

```
suspend connection to data_server.database  
[with nowait]
```

By default, **suspend connection** completes the current transaction before suspending. Use the **with nowait** clause to suspend the connection in mid-transaction. This may be appropriate if a large transaction is responsible for a failure in a replicate database.

Set and Change Parameters Affecting Physical Connections

You set configuration parameters for a connection when you create it. Later, you can update those parameters with Sybase Central, the **alter connection** command.

You can change the configuration of either a single database connection or of all database connections that originate from a single Replication Server. If you are adding many database connections to a Replication Server, you may want to change configuration parameters affecting all connections in order to fine-tune server performance.

To change configuration parameters for all connections originating at the current Replication Server, use the **configure replication server** command.

Configuration parameters that are set for individual connections with **alter connection** override parameters that are set with **configure replication server**. Thus, you can set default parameters with **configure replication server** and then customize settings for specific connections with **alter connection**.

Replication Server provides different types of configuration parameters that affect database connections. There are configuration parameters for:

- Physical database connections.
- Logical database connections. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Alter Warm Standby Database Connections > Alter Logical Connections > Change Parameters Affecting Logical Connections*
- Network-based security.
- Setting up and tuning parallel DSI connections. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads*.

- Tuning Replication Server performance. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance > Connection Parameters that Affect Performance*.

See also

- *Change Parameters Affecting All Connections* on page 200
- *Manage Network-based Security* on page 261

Change Parameters Affecting a Single Connection

After a connection is created, you can change its configuration parameters with the **alter connection** command.

alter connection lets you change the attributes of a database connection. Use this command, for example, if you have added an Adaptive Server database connection using Sybase Central or **rs_init**, and then decide that you want the database connection to use a derived function-string class instead of a system-provided class. The syntax for **alter connection** is:

```
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 {'required' | 'not_required'} |
    set security_services [to] "default"
}
```

You indicate the data server and database that is connected to the Replication Server and specify one or more of the attributes to change. These include:

function_class – the function-string class to use with the database connection.

error_class – the error class to use for handling database errors.

passwd – the new password to use with the login name for the database connection.

log transfer on – allows transactions to be sent, using this connection, to the Replication Server.

log transfer off – stops transactions from being sent, using this connection, from a primary database to the Replication Server.

database_param – updates a configuration parameter that affects connections.

security_param – updates a network security configuration parameter that affects connections.

set security_services [to] 'default' – resets all network-based security features for the connection to “not required.”

*An Example of Using **alter connection***

To change the function-string class for the pubs2 database in the SYDNEY_DS data server to **sqlserver_derived_class**, in the SYDNEY_RS Replication Server enter:

```
suspend connection to SYDNEY_DS.pubs2
alter connection to SYDNEY_DS.pubs2
    set function string to class
        sqlserver_derived_class
resume connection to SYDNEY_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands* for more information about the keywords and options of the **alter connection** command.

See also

- *Manage Network-based Security* on page 261

Change Parameters Affecting All Connections

Use the **configure replication server** command to set default configuration parameters for all connections originating at the source Replication Server.

The syntax for **configure replication server** is:

```
configure replication server
    set database_param to 'value'
```

Configuration changes take effect after you resume the connections.

Changing the dsi_fadeout_time Value

This example shows how to use **configure replication server** to change the **dsi_fadeout_time** parameter so that the DSI connection does not close. Log in to the source Replication Server to execute the commands.

1. Suspend all connections from the source Replication Server. For each connection, enter:

```
suspend connection to data_server.database
```

2. Execute **configure replication server**. Enter:

```
configure replication server
set dsi_fadeout_time to '-1'
```

3. Resume suspended connections from the source Replication Server. For each connection, enter:

```
resume connection to data_server.database
```

Enabling Sybase Failover Support

This example shows how to use **configure Replication Server** to change the **ha_failover** parameter to enable Failover support for all non-RSSD connections from a Replication Server to Adaptive Servers.

1. Execute **configure replication server**. Log in to the Replication Server for which you want to enable Failover support and enter:

```
configure replication server
set ha_failover to 'on'
```

See *Replication Server Administration Guide Volume 2 > Replication System Recovery > Configure the Replication System to Support Sybase Failover*.

Configuration Parameters Affecting Physical Database Connections

Use **alter connection** with the parameter to change the attributes of a single connection or **configure replication server** to change the attribute of the parameter for all connections to Replication Server.

Table 12. Configuration Parameters Affecting Database Connections

Parameter (<i>database_param</i>)	Value (<i>value</i>)
batch	<p>The default, “on,” allows command batches to a replicate database.</p> <p>Default: “on” for ASE and “off” for non-ASE databases.</p> <p>See <i>Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Command Batching for Non-ASE Servers</i>.</p>
batch_begin	<p>Indicates whether a begin transaction can be sent in the same batch as other commands (such as insert and delete).</p> <p>Default: on</p> <p>See <i>Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Command Batching for Non-ASE Servers</i>.</p>
command_retry	<p>The number of times to retry a failed transaction. The value must be greater than or equal to 0.</p> <p>Default: 3</p>
db_packet_size	<p>The maximum size of a network packet. During database communication, the network packet value must be within the range accepted by the database. You may change this value if you have Adaptive Server that has been reconfigured.</p> <p>Maximum: 16384 bytes</p> <p>Default: 512-byte network packet for all Adaptive Server databases</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
deferred_name_resolution	<p>Enable deferred name resolution in Replication Server to support deferred name resolution in Adaptive Server.</p> <p>You must ensure that deferred name resolution is supported in the replicate Adaptive Server before you enable deferred name resolution support in Replication Server.</p> <p>Default: off</p>
disk_affinity	<p>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. Values are “<i>partition_name</i>” and “off.”</p> <p>Default: off</p>
dsi_alt_writetext	<p>Controls how large object updates are sent to the replicate database. Values are:</p> <ul style="list-style-type: none"> • dcany – generates a writetext command that includes primary key columns. This setting prevents full table scans when populating non-ASE replicate databases using Enterprise Connect™ Data Access as an interface. • off (default) – generates an Adaptive Server writetext command that includes a text pointer.
dsi_bulk_copy	<p>Turns the bulk-copy-in feature on or off for a connection. If dynamic_sql and dsi_bulk_copy are both on, Replication Server applies bulk-copy-in when appropriate and uses dynamic SQL if Replication Server cannot use bulk-copy-in. Turn dsi_bulk_copy on to improve performance if you have large batches of inserts.</p> <p>Default: off</p> <hr/> <p>Note: You must set dsi_bulk_copy to off before you enable real-time loading (RTL) replication to Sybase IQ.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_bulk_threshold</p>	<p>The number of consecutive insert commands in a transaction that, when reached, triggers Replication Server to use bulk copy-in. When Stable Queue Transaction (SQT) encounters a large batch of insert commands, it retains in memory the number of insert commands specified to decide whether to apply bulk copy-in. Because these commands are held in memory, Sybase suggests that you do not configure this value much higher than the configuration value for dsi_large_xact_size.</p> <p>Replication Server uses dsi_bulk_threshold for Real-time loading (RTL) replication to Sybase IQ and High volume adaptive replication (HVAR) to Adaptive Server. If the number of commands for an insert, delete, or update operation on one table is less than the number you specify after compilation, RTL and HVAR use language instead of bulk interface.</p> <p>Minimum: 1</p> <p>Default: 20</p> <hr/> <p>Note: You must set dsi_compile_enable to 'on' to use dsi_bulk_threshold.</p>
<p>dsi_charset_convert</p>	<p>The specification for handling character set conversion. This parameter applies to all data and identifiers to be applied at the DSI in question. The values are:</p> <ul style="list-style-type: none"> • “on” (default) – 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.”
<p>dsi_check_lock_wait</p>	<p>The number of milliseconds before the DSI executor thread executes the rs_thread_check_lock function string, which queries the replicate database about lock status.</p> <p>Default: 3000 milliseconds (3 seconds)</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dsi_cmd_batch_size	<p>The maximum number of bytes that Replication Server places into a command batch.</p> <p>Default: 8192 bytes</p>
dsi_cmd_prefetch	<p>Allows DSI to pre-build the next batch of commands while waiting for the response from data server, and therefore improves DSI efficiency. If you also tune your data server to enhance performance, it is likely that you will gain an additional performance increase when you use this feature.</p> <p>Default: off</p> <p>When you set dsi_compile_enable to 'on', Replication Server ignores what you set for dsi_cmd_prefetch.</p>
dsi_cmd_separator	<p>The character that separates commands in a command batch. For example, if you have specified a different separator character and want to change it back to the default character, enter:</p> <pre data-bbox="650 774 1184 852">alter connection to data_server.data-base set dsi_cmd_separator to '<Return>'</pre> <p>Press the Return key, and no other characters, between the two single-quote characters.</p> <p>Default: newline (\n)</p> <hr/> <p>Note: Pressing the Return key is effective only in an interactive update; it is not applicable to executing a script, such as a DDL generated script. You must update this parameter in an interactive mode. You cannot reset it from within a script.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_command_convert</p>	<p>Specifies how to convert a replicate command.</p> <p>A combination of these operations specifies the type of conversion:</p> <ul style="list-style-type: none"> • d – delete • i – insert • u – update • t - truncate • none – no operation <p>Combinations of operations for dsi_command_convert include i2none, u2none, d2none, i2di, t2none, and u2di.</p> <p>You must type the number “2”. The operation before conversion precedes the “2” and the operations after conversion are after the “2”. For example:</p> <ul style="list-style-type: none"> • d2none – do not replicate the delete command. • i2di,u2di – convert both insert and update to delete followed by insert, which is equivalent to an auto-correction. • t2none – do not replicate truncate table command. <p>Default: None.</p> <p>You can also configure this parameter at the table-level.</p> <p>For setting, use alter connection for database-level, or alter connection with the for replicate table named clause for table-level configuration.</p>
<p>dsi_commit_check_locks_intrvl</p>	<p>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. See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads</i>.</p> <p>Default: 1000 ms (1 second)</p> <p>Minimum: 0</p> <p>Maximum: 86,400,000 ms (24 hours)</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dsi_commit_check_locks_max	<p>The maximum number of times a DSI executor thread checks whether it is blocking other transactions in the replication database before rolling back its transaction and retrying it. Used with parallel DSI. See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads</i>.</p> <p>Default: 400</p> <p>Minimum: 1</p> <p>Maximum: 1,000,000</p>
dsi_commit_control	<p>Specifies whether commit control processing is handled internally by Replication Server using internal tables (on) or externally using the <code>rs_threads</code> system table (off). Used with parallel DSI. See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads</i>.</p> <p>Default: on</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_compile_enable</p>	<p>Enables or disables RTL or HVAR at the server-level, database-level, or table-level.</p> <p>For setting, use configure replication server for server-level, alter connection for database-level, or alter connection with the for replicate table named clause for table-level configuration.</p> <p>Default:</p> <ul style="list-style-type: none"> • off – server and database-level. Replication Server uses continuous log order row by row change replication. • on – table-level <p>Set dsi_compile_enable to 'off' for an affected table if replicating new row changes causes problems, such as when there is a trigger on the table which requires all the operations on that table to be replicated in log order, and therefore compilation is not allowed.</p> <hr/> <p>Note: Set dsi_compile_enable to 'on' at the server- or database-level before you set dsi_compile_enable to 'on' at the table-level.</p> <hr/> <p>When you set dsi_compile_enable to 'on', Replication Server ignores what you set for replicate_minimal_columns and dsi_cmd_prefetch.</p> <p>After you execute dsi_compile_enable at the server, database, or table-level, suspend and resume the connection.</p> <p>See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Advanced Services Option > High Volume Adaptive Replication to Adaptive Server</i> for HVAR.</p> <p>See <i>Replication Server Heterogeneous Replication Guide > Sybase IQ as Replicate Data Server</i> for RTL.</p> <p>License: Separately licensed under the Advanced Services Option. See the <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Advanced Services Option</i>.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_compile_max_cmds</p>	<p>Specifies, in number of commands, the maximum size of a group of transactions. When HVAR or RTL reaches the maximum group size for the current group that it is compiling, HVAR or RTL starts a new group.</p> <p>If there is no more data to read, and even if the group does not reach the maximum number of commands, HVAR or RTL completes grouping the current set of transactions into the current group.</p> <p>For setting, use configure replication server for server-level or alter connection for database-level.</p> <p>Minimum: 100</p> <p>Default: 10,000</p> <hr/> <p>Note: You must set dsi_compile_enable to 'on' to use dsi_compile_max_cmds.</p>
<p>dsi_dataserver_make</p>	<p>Specifies the data server type that contains the replicate database that you want to connect to.</p> <p>Set to:</p> <ul style="list-style-type: none"> • ase – to replicate to Adaptive Server • ase – to replicate to IBM DB2 on z/OS • hdb – to replicate to HANA • iq – to replicate to Sybase IQ • ase – to replicate to Microsoft SQL Server • ora – to replicate to Oracle • udb – to replicate to IBM UDB <p>Use dsi_dataserver_make and dsi_connector_type to identify the connector that is associated with the connection.</p> <p>You can configure dsi_dataserver_make at the database level.</p> <p>If you do not specify this parameter, Replication Server deduces the data server type from the function-string class name of the database connection.</p> <p>If the functions-string class is customized, Replication Server cannot deduce the data server type and defaults to 'ASE'.</p>
<p>dsi_exec_request_sproc</p>	<p>Turns on or off request stored procedures at the DSI of the primary Replication Server.</p> <p>Default: on</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dsi_fadeout_time	<p>The number of seconds of idle time before a DSI connection is closed. A value of -1 specifies that the connection should not fade out.</p> <p>Default: 600 seconds</p>
dsi_ignore_underscore_name	<p>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.”</p> <p>Default: on</p>
dsi_isolation_level	<p>Specifies the isolation level for transactions. ANSI standard and Adaptive Server supported values are:</p> <ul style="list-style-type: none"> • 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, prevents dirty reads, and ensures that data written by one transaction represents the actual data. • 3 – prevents phantom rows, prevents nonrepeatable reads, prevents dirty reads, and ensures that data written by one transaction represents the actual data. <p>Through the use of custom function strings, Replication Server can support any isolation level the replicate data servers may use. Support is not limited to ANSI standard only.</p> <p>The default value is the current transaction isolation level for the target data server.</p>
dsi_keep_triggers	<p>Specifies whether triggers should fire for replicated transactions in the database.</p> <p>“off” – causes Replication Server to set triggers off in the Adaptive Server database, so that triggers do not fire when transactions are executed on the connection.</p> <p>Use this setting for standby databases.</p> <p>on” – specifies all databases except standby databases.</p> <p>Default: on (except standby databases)</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dsi_large_xact_size	<p>The number of commands allowed in a transaction before the transaction is considered to be large.</p> <p>Default: 100</p> <p>Minimum: 4</p> <p>Maximum: 2,147,483,647</p> <p>Replication Server ignores dsi_large_xact_size when you turn on dsi_compile_enable.</p>
dsi_max_cmds_in_batch	<p>Defines maximum number of source commands for which output commands can be batched.</p> <p>You must suspend and resume the connection for any change in the parameter to take effect.</p> <p>Range: 1 – 1000</p> <p>Default: 100</p>
dsi_max_cmds_to_log	<p>The number of commands to write into the exceptions log for a transaction.</p> <p>Default: -1 (all commands)</p> <p>Valid values: 0 to 2147483647</p>
dsi_max_xacts_in_group	<p>Specifies the maximum number of transactions in a group. Larger numbers may improve data latency at the replicate database. Range of values: 1 – 1000.</p> <p>Default: 20</p> <p>This parameter is ignored when dsi_compile_enable is turned on.</p>
dsi_max_text_to_log	<p>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, or image columns from filling the RSSD or its log.</p> <p>Default: -1 (all text, unitext, or image columns)</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dsi_non_blocking_commit	<p>Specifies the number of minutes to extend the period of time Replication Server saves messages after a commit. Range of values: 0 – 60 minutes.</p> <p>Default: 0 – non-blocking commit is disabled.</p> <p>Enable this parameter to improve replication performance when the delayed commit feature is available in Adaptive Server 15.0 and later or the equivalent feature is available in Oracle 10g v2.</p>
dsi_num_large_xact_threads	<p>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.</p> <p>Default: 0</p>
dsi_num_threads	<p>The number of parallel DSI threads to be used. The maximum value is 255.</p> <p>Default: 1</p>
dsi_partitioning_rule	<p>Specifies the partitioning rules (one or more) the DSI uses to partition transactions among available parallel DSI threads. Values are origin, origin_sessid, time, user, name, and none.</p> <p>See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads > Partitioning Rules: Reducing Contention and Increasing Parallelism</i> for detailed information.</p> <p>Default: none</p> <p>This parameter is ignored when dsi_compile_enable is turned on.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_proc_as_rpc</p>	<p>Specifies how Replication Server applies stored procedure replication.</p> <ul style="list-style-type: none"> • Set on to use remote procedure call (RPC) calls. • Set off to use language calls. <p>Default: off</p> <p>When the replicate database is Adaptive Server, dsi_proc_as_rpc can be on or off.</p> <p>When the replicate database is Oracle, set dsi_proc_as_rpc on if you use ExpressConnect for Oracle (ECO). ECO only supports stored procedure replication using RPC. By default, Replication Server sets dsi_proc_as_rpc on if you use one of the Oracle ECO connection profiles when you create the connection to the Oracle database from Replication Server. See <i>Replication Server Options > ExpressConnect for Oracle Installation and Configuration Guide ExpressConnect for Oracle Microsoft Windows, UNIX, and Linux > Configuring ExpressConnect for Oracle</i>.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_quoted_identifier</p>	<p>Enables or disables quoted identifier support in the Data Server Interface (DSI).</p> <ul style="list-style-type: none"> • on – enables quoted identifiers if you mark the table as quoted in a replication definition or if the LTL that is sent by RepAgent from Adaptive Server 15.7 marks the table as quoted • off – disable quoted identifier support • always – always surrounds identifiers with double quotes, regardless of primary database configuration or replication definition setting. You can use the always option for a specific table or for all tables. <p>Default: off</p> <p>If you set dsi_quoted_identifier to on for an Adaptive Server database connection , you must not include double quotes in any stored procedure you want to replicate. Otherwise, the DSI thread shuts down.</p> <p>Enable this parameter to:</p> <ul style="list-style-type: none"> • Create or modify a connection that allows quoted identifiers to be forwarded to data servers. Use create connection or alter connection to set dsi_quoted_identifier. Use admin config to check the value of dsi_quoted_identifier. • Support marking of identifiers in a replication definition as quoted. <p>Quoted identifiers are not supported in mixed version environments. For replication of a quoted identifier to succeed, the primary Replication Server and the Replication Server that connects to the replicate data server version must be 15.2 and later. However, intermediate Replication Servers in a route can be of lower versions.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_replication</p>	<p>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, usually not replicated further (except if there is a standby database), setting this parameter to “off” avoids writing unnecessary information into the transaction log.</p> <p>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.</p> <p>Default: on (“off” for standby database in a warm standby application)</p>
<p>dsi_row_count_validation</p>	<p>If you have table rows that are not synchronized, and you want to bypass the default error actions and messages, you can set dsi_row_count_validation to off to disable row count validation.</p> <p>Default: on to enable row count validation.</p> <p>You need not suspend and resume a database connection when you set dsi_row_count_validation for the connection; the parameter takes effect immediately. However, the new setting affects the batch of replicated objects that Replication Server processes after you execute the command. Changing the setting does not affect the batch of replicated objects that Replication Server is currently processing.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_serialization_method</p>	<p>Specifies the method used to determine when a transaction can start, while still maintaining consistency. In all cases, commit order is preserved.</p> <p>These 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.</p> <ul style="list-style-type: none"> • 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. • wait_after_commit – specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it has committed completely. <hr/> <p>Note: You can only set dsi_serialization_method to no_wait if dsi_commit_control is set to “on”.</p> <hr/> <p>These options are retained only for backward compatibility with older versions of Replication Server:</p> <ul style="list-style-type: none"> • none – same as wait_for_start. • single_transaction_per_origin – same as wait_for_start with dsi_partitioning_rule set to origin. <hr/> <p>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.</p> <hr/> <p>Default: wait_for_commit</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_sqt_max_cache_size</p>	<p>Maximum SQT (Stable Queue Transaction interface) cache memory for the database connection, in bytes.</p> <p>The default, “0,” means that the current setting of sqt_max_cache_size is used as the maximum cache size for the connection.</p> <p>Default: 0</p> <p>For 32-bit Replication Server:</p> <ul style="list-style-type: none"> • Minimum – 0 • Maximum – 2147483647 <p>For 64-bit Replication Server:</p> <ul style="list-style-type: none"> • Minimum – 0 • Maximum – 2251799813685247
<p>dsi_stage_all_ops</p>	<p>Prevents compilation for specified tables when you configure Replication Server and Sybase IQ InfoPrimer integration.</p> <p>If table history must be preserved, as in the case of slowly changing dimension (SCD) tables, set dsi_stage_all_ops to on.</p> <p>See <i>Replication Server Heterogeneous Replication Guide > Sybase IQ as Replicate Data Server > Replication Server and Sybase IQ InfoPrimer Integration > Parameters > dsi_stage_all_ops</i>.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_text_convert_multiplier</p>	<p>Changes the length of <code>text</code> or <code>unitext</code> datatype columns at the replicate site. Use dsi_text_convert_multiplier when <code>text</code> or <code>unitext</code> datatype columns must expand or contract due to character set conversion. Replication Server multiplies the length of primary <code>text</code> or <code>unitext</code> data by the value of dsi_text_convert_multiplier to determine the length of <code>text</code> or <code>unitext</code> data at the replicate site. The value type is float.</p> <ul style="list-style-type: none"> • If the character set conversion involves expanding <code>text</code> or <code>unitext</code> datatype columns, set dsi_text_convert_multiplier equal to or greater than 1.0. • If the character set conversion involves contracting <code>text</code> or <code>unitext</code> datatype columns, set dsi_text_convert_multiplier equal to or less than 1.0. <p>Default: 1</p>
<p>dsi_timer</p>	<p>Use the dsi_timer configuration parameter to specify a delay between the time transactions commit at the primary database and the time transactions commit at the standby or replicate database. Replication Server processes transactions in the outbound queue in commit order after the period of delay is over.</p> <p>After you execute dsi_timer with alter connection or alter logical connection, suspend and resume the connection.</p> <p>Specify the delay in the hh:mm format.</p> <ul style="list-style-type: none"> • Maximum: 24 hours. • Default: 00:00, which means there is no delay. <p>Note: Replication Server does not support time zone differences between the Replication Agent at the primary database and the Replication Server with the DSI connection where you want to execute dsi_timer.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_top1_enable</p>	<p>For Adaptive Server databases, set dsi_top1_enable to on to enable replication of tables that do not have unique keys.</p> <p>If a table does not have a unique key, it is possible that there are two or more rows with the same values. However, DSI shuts down if it cannot find a unique row to apply an operation. The dsi_top1_enable parameter instructs DSI to select and update only the first instance of multiple similar rows by setting <i>unsigned_integer</i> to 1 in the top unsigned_integer clause of the Adaptive Server select statement.</p> <p>Default: off</p> <p>Use configure replication server to set the parameter for all replicate connections. Do not use configure replication server to set dsi_top1_enable to on if there are connections to non-Adaptive Server replicate databases. Instead, use alter connection to set the parameter individually for each Adaptive Server replicate database connection.</p> <p>Replication Server does not support replication of a table without unique keys if:</p> <ul style="list-style-type: none"> • There is a customized function string for the table. • There is a table replication definition for the table. • Transactions use updates and deletes with the like option in the where clause at the primary database. • The replicate database is not Adaptive Server.

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>dsi_xact_group_size</p>	<p>The maximum number of bytes, including stable queue overhead, to place into one grouped transaction. A grouped transaction is a set of transactions that the DSI applies as a single transaction. A value of -1 means no grouping.</p> <p>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.</p> <hr/> <p>Note: Obsolete for Replication Server version 15.0 and later. Retained for compatibility with older Replication Servers.</p> <hr/> <p>Maximum: 2,147,483,647</p> <p>Default: 65,536 bytes</p> <p>This parameter is ignored when dsi_compile_enable is turned on.</p>
<p>dump_load</p>	<p>Set to “on” at replicate sites only to enable coordinated dump. See <i>Replication Server Administration Guide Volume 2 > Replication System Recovery > Recover from Primary Database Failures > Loading from Coordinated Dumps</i>.</p> <p>Default: off</p>
<p>exec_cmds_per_timeslice</p>	<p>Specifies the number of LTL commands an LTI or RepAgent Executor thread can process before it must yield the CPU to other threads. The range is 1 to 2,147,483,647.</p> <p>Default: 2,147,483,647</p>
<p>dynamic_sql</p>	<p>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.</p> <hr/> <p>Note: If dynamic_sql and dsi_bulk_copy are both on, DSI applies bulk copy-in. Dynamic SQL is used if bulk copy-in is not used.</p> <hr/> <p>Default: off</p> <hr/> <p>Note: You must set dynamic_sql to off before you enable real-time loading (RTL) replication to Sybase IQ.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
dynamic_sql_cache_size	<p>Gives the Replication Server a hint on how many database objects may use the dynamic SQL statement for a connection. Minimum: 1</p> <p>Maximum: 65536</p> <p>Default: 100</p>
dynamic_sql_cache_management	<p>Manages the dynamic SQL cache for a DSI/E thread. Values: mru - keep the most recently used statements and deallocate the to allocate new dynamic statements when dynamic_sql_cache_size is reached. fixed (default)- Replication Server stops allocating the new dynamic statements once dynamic_sql_cache_size is reached.</p>
exec_nrm_request_limit	<p>Specifies the amount of memory available for messages from a primary database waiting to be normalized.</p> <p>Set nrm_thread to 'on' with configure replication server before you use exec_nrm_request_limit.</p> <p>Minimum: 16,384 bytes</p> <p>Maximum: 2,147,483,647 bytes</p> <p>Default for:</p> <ul style="list-style-type: none"> • 32-bit – 1,048,576 bytes (1MB) • 64-bit – 8,388,608 bytes (8MB) <p>After you change the configuration for exec_nrm_request_limit, suspend and resume the Replication Agent.</p> <p>License: Separately licensed under the Advanced Services Option. See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Advanced Services Option</i> .</p>
exec_sqm_write_request_limit	<p>Specifies the amount of memory available for messages waiting to be written to an inbound queue.</p> <p>Default: 1MB</p> <p>Minimum: 16KB</p> <p>Maximum: 2GB</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
<p>md_sqm_write_request_limit</p>	<p>Specifies the amount of memory available to the Distributor for messages waiting to be written to the out-bound queue.</p> <hr/> <p>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.</p> <hr/> <p>Default: 1MB Minimum: 16K Maximum: 2GB</p>
<p>rep_as_standby</p>	<p>When rep_as_standby is on, table subscriptions replicate tables marked by sp_reptostandby.</p> <p>For rep_as_standby on to succeed, the RepAgent parameters send maint xacts to replicate must be false and send warm standby xacts must be true.</p> <p>Default: off</p>
<p>replicate_minimal_columns</p>	<p>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 replicate database.</p> <p>Values are On and Off.</p> <p>Default: On</p> <p>Replication Server uses this connection-level parameter when a replication definition does not contain the replicate minimal columns clause, or if there is no replication definition at all</p> <p>Otherwise, Replication Server ignores the value of this parameter.</p> <p>You can use admin config to display replicate_minimal_columns configuration information.</p> <p>When you set dsi_compile_enable to 'on', Replication Server ignores what you set for replicate_minimal_columns.</p> <p>See <i>Replication Server Administration Guide Volume 2 > Performance Tuning > Dynamic SQL for Enhanced Replication Server Performance > Use Replicate Minimal Columns with Dynamic SQL</i>.</p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
save_interval	<p>The number of minutes that the Replication Server saves messages after they have been successfully passed to the destination data server.</p> <p>Default: 0 minutes</p>
stage_operations	<p>Set to on for Relication Server to write operations to staging tables for the specified connection when you configure Replication Server and Sybase IQ InfoPrimer integration.</p> <p><i>See Replication Server Heterogeneous Replication Guide > Sybase IQ as Replicate Data Server > Replication Server and Sybase IQ InfoPrimer Integration > Parameters > stage_operations.</i></p>
sub_sqm_write_request_limit	<p>Specifies the memory available to the subscription materialization or dematerialization thread for messages waiting to be written to the outbound queue.</p> <p>Default: 1MB</p> <p>Minimum: 16K</p> <p>Maximum: 2GB</p>
unicode_format	<p>Supports sending unicode data in U&'' format.</p> <p>Set unicode_format to one of these values:</p> <ul style="list-style-type: none"> • string – unicode characters are converted to character string format. For example, the <code>string</code> "hello" is sent out as "hello". • ase – unicode characters are sent out in U&' ' format. For example, the <code>string</code> "hello" is sent out as "U&\0068\0065\006c\006c\006f". The two-byte unicode value is sent in network order as required by Adaptive Server Enterprise. <p>Default: <code>string</code></p>

Parameter (<i>database_param</i>)	Value (<i>value</i>)
use_batch_markers	<p>If use_batch_markers is set to on, the function strings rs_batch_start and rs_batch_end will be executed.</p> <hr/> <p>Note: This parameter must be set to on only for replicate data servers that require additional SQL translation to be sent at the beginning and end of a batch of commands that are not contained in the rs_begin and rs_commit function strings.</p> <hr/> <p>Default: off</p> <p>See <i>Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Command Batching for Non-ASE Servers</i>.</p>

Change Replication Server Connection Parameters to Improve Performance

You can use configuration parameters to improve replication performance.

Sybase sets default values for configuration parameters for average installations and usage. Depending on your system configuration and how Replication Server is used at your site, you may find that careful altering of certain default values may improve performance. See *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance* for a general discussion of performance and configuration parameters. See also *Replication Server Administration Guide Volume 2 > Performance Tuning > Use Parallel DSI Threads*.

If you are adding many new connections, you may want to change the **memory_limit** or **num_threads** Replication Server parameters to improve performance.

See *Replication Server Reference Manual > Replication Server Commands > **configure replication server*** for more information about the **memory_limit** and **num_threads** parameters.

Increasing memory_limit

To increase the amount of memory specified for Replication Server, increase the value specified for the **memory_limit** parameter by using **configure replication server** at Replication Server.

For example, execute **configure replication server** in the following manner to increase **memory_limit** to 25MB:

```
configure replication server
    set memory_limit to '25'
```

Increasing num_threads

You may need to increase the number of Open Server threads that the Replication Server can use. To do this, increase the value specified for the **num_threads** parameter, using **configure replication server** at the Replication Server.

For example, execute **configure replication server** in the following manner to increase **num_threads** to 70:

```
configure replication server
    set num_threads to '70'
```

Resume Database Connections

After you have changed the attributes of a database connection, you can resume activity on the connection either in Sybase Central or by using the **resume connection** command.

To resume a database connection from the command line, enter:

```
resume connection to data_server.database
[skip [n] transaction | execute transaction]
```

When the connection is resumed, Replication Server retrieves rows from the `rs_lastcommit` system table so that it can find the correct place in the transaction stream to begin submitting transactions.

The optional **skip [n] transaction** clause instructs Replication Server skip a specified number of transactions in the connection queue before resuming the connection. The first *n* transactions are written to the exceptions log.

The **skip [n] transaction** clause is necessary if the first *n* transactions cause Replication Server to suspend the connection and the cause of the failure cannot be corrected. For example, the transaction may have produced a data server error that is assigned the **retry_stop** or **stop_replication** error action. Or, perhaps it was necessary to use **suspend connection** and the **with nowait** clause to manually interrupt the transaction.

Warning! If you execute **resume connection** with the **skip transaction** clause, you must correct any inconsistency that results from the lost transaction. Only use the **skip transaction** clause when the condition causing the transaction to fail cannot be corrected.

The optional **execute transaction** clause instructs Replication Server to execute the first transaction in the connection's queue. Use this clause only when a system transaction has failed to execute. See *Replication Server Administration Guide Volume 2 > Errors and Exceptions Handling > Duplicate Detection for System Transactions* for information about error handling for system transactions.

Changing Replicate Databases to Primary Databases

Each primary database must have a Replication Agent that scans the database log and transfers data to the Replication Server for distribution to replicate databases. If you want to change an

Adaptive Server database that is designated as replicate-only to be a source of replicated functions or to contain primary data, you must enable the RepAgent thread for the database.

1. Configuring RepAgent at the Replication Server

- a) Create a RepAgent user so that RepAgent can log in to Replication Server. Use the **create user** command where *ra_user_name* is the name of the RepAgent user and *ra_password* is the RepAgent password:

```
create user ra_user_name
set password {ra_password | null}
```

- b) Grant this user **connect source** permission, using the **grant** command:

```
grant connect source to ra_user_name
```

If the Replication Server already manages a primary database, you can use the “RepAgent user” that already exists for the new primary database.

- c) Execute **alter connection** using the **log transfer on** option:

```
alter connection to data_server.database
set log transfer to 'on'
```

2. Configuring RepAgent at the Adaptive Server database

- a) If the name of the Adaptive Server has not yet been defined, you must define it using the following command where *lname* is the Adaptive Server name:

```
sp_addserver lname, local
```

- b) If RepAgent threads have not been enabled for the Adaptive Server, you must enable them:

```
sp_configure 'enable rep agent threads'
```

- c) Configure RepAgent for the database with **sp_config_rep_agent**:

```
sp_config_rep_agent dbname, 'enable', 'rs_name',
'rs_user_name', 'rs_password'
```

Note: The value for *rs_user_name* and *rs_password* configured at the Adaptive Server must be the same for *ra_user_name* and *ra_password* created at the Replication Server in step 1.

- d) Create the **rs_marker** stored procedure and set its replicate status to “true”, using the **sp_setreplicate** system procedure.

You can find the **rs_marker** stored procedure in the file *rs_install_primary.sql* or *rsinssys.sql* in the *scripts* directory of the Sybase release directory.

- e) Start RepAgent:

```
sp_start_rep_agent dbname
```

See also

- *Configuring RepAgent* on page 113

Create the rs_marker Stored Procedure

Create the **rs_marker** stored procedure to check on replication status.

Replication Server executes the **rs_marker** system function in a primary database during subscription materialization. The function works by executing a replicated stored procedure that is also named **rs_marker**. The procedure checks to make sure it is marked replicated and issues a warning if it is not. Because the **rs_marker** stored procedure is replicated, Adaptive Server records its executions in the transaction log for the database, where they can be read by RepAgent.

Sybase Central and **rs_init** create **rs_marker** when you designate a database as having primary data. It is not required in databases that have no primary data. The exact text of the stored procedure can always be found in `rs_install_primary.sql` or `rsinssys.sql` in the `scripts` directory of the Sybase release directory.

Here is a sample text:

```
create procedure rs_marker
    @rs_api varchar(255)
as
    declare @rep_constant smallint
    select @rep_constant = -32768
    if not exists (select sysstat from sysobjects
        where name = 'rs_marker'
        and type = 'P'
        and sysstat & @rep_constant != 0)
    begin
        print "Have your DBO execute
        'sp_setreplicate' on the procedure
        'rs_marker'"
    return(1)
    end
```

rs_marker does not modify data in the database. Its purpose is to execute so that it can be recorded in the transaction log.

If **rs_marker** is not marked as replicated, you can mark it with **sp_setreplicate**:

```
sp_setreplicate rs_marker, 'true'
```

Changing Primary Databases to Replicate Databases

You can change the primary database to a replicate database.

At the:

1. Current replicate Replication Server.
 - a) Drop all subscriptions and publication subscriptions to the replication definitions in this database.
2. Current primary Replication Server.
 - a) Drop all replication definitions defined for this database.

3. Adaptive Server.

- a) Shut down RepAgent:

```
sp_stop_rep_agent dbname
```

- b) Disable RepAgent:

```
sp_config_rep_agent dbname, disable
```

4. Replication Server

- a) Log in to the Replication Server that manages the database and execute
- alter connection**
- using the
- log transfer off**
- option:

```
alter connection to data_server.database
set log transfer off
```

5. Adaptive Server

- a) Set the status of
- rs_marker**
- to “false:”

```
sp_setreplicate rs_marker, 'false'
```

- b) Set the replicate status of all replicated objects to “false”:

1. Execute **sp_setreptable** without arguments to generate a list of all replicated tables and stored procedures in the database.
2. One by one, set the replicate status of each table and stored procedure to “false,” using **sp_setreptable** and **sp_setrepproc**.

Drop Database Connections

To remove a database from the replication system, use Sybase Central or execute **drop connection**.

Before you execute the command, drop any subscriptions for replication definitions for data in the database. If you are dropping a connection to a primary database, first drop all replication definitions for tables in the database.

Note: **drop connection** removes database connection information from the Replication Server system tables. It does not remove replicate data from any database in the system. To remove replicate data, use **drop subscription** using the **with purge** option.

To drop a connection, specify the data server with the database whose connection is to be dropped. The syntax is:

```
drop connection to data_server.database
```

For example, to drop the connection to the `pubs2` database in the `SYDNEY_DS` data server, enter:

```
drop connection to SYDNEY_DS.pubs2
```

Note: If you are using RepAgent for log transfer, you should also stop (if necessary) and then disable RepAgent at the primary database.

Manage Database Connections

For information about dropping logical connections, see *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Alter Warm Standby Database Connections > Drop Logical Database Connections*.

See also

- *Disable RepAgent* on page 116

Drop a Database from the ID Server

Use **sysadmin dropdb** to drop a database from the ID Server.

Replication system databases, data servers, and Replication Servers are listed in the `rs_idnames` system table in the RSSD for the ID Server. Occasionally, you may need to remove the entry for a database from this system table.

For example, the **drop connection** command fails and you want to reuse the connection name. You must force the ID Server to delete from the `rs_idnames` system table the row that corresponds to the database. (Physical database connections have a “P” in the `ltype` column in this system table.)

Log in to the ID Server and execute the **sysadmin dropdb** command to delete the entry for the specified database. The syntax for **sysadmin dropdb** is:

```
sysadmin dropdb, data_server, database
```

You must have **sa** permission to execute any **sysadmin** command.

Monitor Database Connections

Use Sybase Central, or stored procedures and RCL commands to monitor database connections.

See *Replication Server Troubleshooting Guide* if you need to monitor connections for troubleshooting purposes.

See also

- *Manage a Replication System* on page 83

View Current Database Connections

To check the status of current database connections, use Sybase Central or **admin show connections**.

admin show_connections displays information about all database connections from the Replication Server. **admin show_connections** also displays information about all routes from the Replication Server.

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

List Databases Managed by a Replication Server

To list the databases that a Replication Server manages, use Sybase Central or the **rs_helpdb** stored procedure at the Replication Server RSSD.

The `rs_databases` system table contains entries for all of the databases managed by the Replication Server, including databases managed by other Replication Servers that have a route to the Replication Server.

The syntax for **rs_helpdb** is:

```
rs_helpdb [data_server, database]
```

See *Replication Server Reference Manual > RSSD Stored Procedures > rs_helpdb* for detailed usage and syntax information .

Display DSI Thread Status

To view DSI thread status, use Sybase Central or the **admin who** commands to display thread status information.

- **admin who** displays all threads in the system, including DSI threads.
- **admin who, dsi** displays the status of the DSI thread, which Replication Server starts to submit transactions to the data server.

See the *Replication Server Reference Manual > Replication Server Commands > admin who* for complete thread status listings.

See also

- *Manage a Replication System* on page 83

Manage Replication Server Security

Careful management of login names, passwords, and permissions is essential to the security of the replication system.

Replication Server login names and specific permissions are required for:

- Each component of the replication system, such as the data server and the Replication Server.
With Replication Server 15.2, you can use Replication Server gateway which allows to access components in the replication system with a single user name and password.
- Each user who is setting up replicated data or monitoring and managing the Replication Server.

You can set up encrypted passwords throughout the replication system and change passwords that are encrypted. Refer to the Replication Server installation and configuration guides for your platform for details on password encryption.

You can use RCL commands to manage Replication Server security, including creating and modifying login names, passwords, and permissions, and the dependencies involved in making modifications. In addition, Replication Server supports third-party security services that ensure secure message transmission over the network and enable user authentication for seamless login to Replication Servers in the replication system.

See also

- *Manage Network-based Security* on page 261
- *Starting Sybase Central* on page 61

Manage Replication Server System Security

You must establish login names and passwords for the various components of the Replication Server system, including RSSDs, RepAgent, the ID Server, and Replication Server.

Often, one process must log in to another, remote process. In such cases, the login name and password assigned to the process logging in must also exist at the remote process. If a password used to log in to the remote process is changed at the current process only, login attempts fail.

As a general rule, if you are specifying or modifying system login names, keep the names unique. If you use the same login name for different roles, then any time you change the password, many of the dependencies described in this section are affected.

Replication System Login Names

There are several login names required for replication system components.

Table 13. Overview of Replication System Login Names

Source Server	Destination Server and Database	Login Name Description
Primary Replication Server	Primary Adaptive Server and RSSD	RSSD primary user
Replicate Replication Server	Replicate Adaptive Server and RSSD	RSSD maintenance user
Replicate Replication Server	Replicate Adaptive Server and replicate database	database maintenance user
RepAgent for RSSD	Replication Server	RepAgent user for RSSD
RepAgent for primary database	Replication Server	RepAgent user for primary database
Replication Servers	ID Server (Replication Server)	ID Server user
Replication Servers	Other Replication Servers	Replication Server user (RSI user)

RSSD Login Names and Passwords

Replication Server uses the primary user and maintenance user login names to manage security.

When you install Replication Server, the **rs_init** program creates the primary and maintenance Adaptive Server login names to maintain the RSSD.

Replication Server uses the primary user login name to modify the system tables in the RSSD for the primary Replication Server. Modifications may include route, replication definition, and function-string information changes to be replicated to RSSDs for other Replication Servers. You set up the primary user when you create the primary RSSD using **rs_init**.

Replication Server uses the maintenance user login name to apply modifications to replicate RSSDs. RepAgent filters out RSSD modifications made by the maintenance user to avoid replicating them to other RSSDs. You set up the maintenance user when you create the replicate RSSD using **rs_init**.

If the login name or password is changed for either the primary or maintenance user, edit the Replication Server configuration file to match these changes, and restart the Replication Server.

Guidelines for Changing RSSD Primary User Login Name and Password

Observe the guidelines when you change the RSSD primary user login name and password.

- Never change the RSSD primary user login name and/or password while routes are being created.

While a route is being created, the destination Replication Server uses the primary user login name and password to create and materialize subscriptions at the destination site for replicated RSSD system tables.

- Be sure to also apply the same RSSD primary user login name and/or password changes to the Replication Server.
 - To change an encrypted or clear text password, use **alter user** with the **set password** clause.
 - To change both a login name and password (encrypted or clear text), use **drop user** to drop the old login name and **create user** to create the new login name and password. Then grant the user **primary subscribe** permission.
 - Update the Replication Server configuration file with the new login name and/or password. Use **rs_init** if the password is encrypted.
 - For the updates to take effect, restart the Replication Server.

See *Replication Server Reference Manual* > *Replication Server Commands* > **alter user** for command syntax details.

See also

- *Manage Replication Server Permissions* on page 251
- *Replication Server Permissions* on page 251

Replication Server Login Name and Password for RepAgent

Learn the guidelines, commands, and procedures to change the Replication Server login name and password for RepAgent.

RepAgent retrieves information about changes to the replicated system tables in the RSSD or to the primary database from the database transaction logs and submits them to the Replication Server for distribution.

Replication Server needs a login name for RepAgent. The **rs_init** program uses the **create user** command to add this Replication Server user.

Observe these guidelines when you change the Replication Server login name and/or password for the RepAgent. The login name and password you create at the Replication Server must be the same as that used to configure the RepAgent at Adaptive Server.

At Replication Server:

- To change the password, use the **alter user** command with the **set password** clause.

Manage Replication Server Security

- To change both the login name and password, use the **drop user** command to drop the old user login name and the **create user** command to create the new login and password. Then grant the user **connect source** permission.

At Adaptive Server:

- To change the login name and password, use the **sp_config_rep_agent** system procedure with the *dbname*, *rs_servername*, *rs_username*, and *rs_password* options. This updates the login name and password in the database `sysattributes` table. The password is always encrypted.
- For the updates to take effect, restart RepAgent.

See *Replication Server Reference Manual > Replication Server Commands > alter user* and, *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_config_rep_agent* for syntax details.

ID Server Login Name and Password

Learn the guidelines to change the ID Server login name and password.

The ID Server registers Replication Servers and databases in a replication domain. Replication Servers use the *ID_user* configuration parameter in the Replication Server configuration file to connect to the ID Server. For each Replication Server, the ID Server login name and password must match the ID Server entry.

The ID Server must be the first Replication Server installed. The ID Server login name and password are established using **rs_init**.

If you change the login name and/or password for the ID Server, be sure to modify *ID_user* in the Replication Server configuration file of each Replication Server that is defined to the ID Server, as well as the Replication Server configuration file for the ID Server itself. You can make password changes using **rs_init**.

You also must change the ID Server login name and/or password in the Replication Server.

See also

- *Manage Replication Server Login Names and Passwords* on page 243

Replication Server Login Name and Password for Replication Servers

Replication Servers log in to other Replication Servers to send operations.

The login name is created using **rs_init**. The login name is used when a direct route is created, from one Replication Server to another.

To change the password for a login name used for a direct route, execute the **alter route** command.

See also

- *Manage Routes* on page 165

Maintenance User Adaptive Server Login Name and Password

Replication Servers log in to Adaptive Server for the RSSD database or a user database using the maintenance user login name. When applying **insert**, **delete**, or **update** primary operations to replicate databases, Replication Server uses the maintenance user login name and password.

Note: Among the permissions granted to the maintenance user is **replication_role**. If you revoke maintenance user's **replication_role**, Replication Server will not replicate **truncate table** unless the maintenance user has been granted **sa_role**, owns the table, or is aliased as the database owner.

To change the password for the maintenance user, use the **alter connection** command.

Maintenance User Security

To increase security, prevent unauthorized access to the database, and restrict maintenance user access to Replication Server, secure the maintenance user with password encryption, and set an expiration interval for the password.

Replication Server defines the maintenance user ID and password with the Replication Server **create connection** command or system management tools such as **rs_init** that you use for creating a connection to the replicate database.

Replication Server logs in to the replicate data server as the maintenance user for replication. The maintenance user must have the permissions to execute replicated stored procedures, and to execute DML operations such as **insert**, **delete**, and **update** on replicate tables. To enable replication of DDL operations such as **create table** using the maintenance user, the maintenance user must have **set session authorization** permission to perform DDL operation as the original user.

Set **hide_maintuser_pwd** on to:

- Periodically generate a new password for the maintenance user for existing connections to a replicate Adaptive Server database.
- Generate and encrypt the password for any new database connections you create, and periodically regenerate and encrypt the password at the replicate Adaptive Server database.
- Change and encrypt any password you reenter for the maintenance user.

Use **maintuser_pwd_expiration** to set the password expiration interval for the maintenance user.

Maintenance User Password Protection

Use `hide_maintuser_pwd` to configure password protection for the maintenance user and restrict maintenance user access to Replication Server.

You can configure `hide_maintuser_pwd` for a specific Adaptive Server database connection controlled by the Replication Server or at the server level for all Adaptive Server database connections controlled by the Replication Server. To configure `hide_maintuser_pwd` at the:

- Connection level – use **create connection** or **alter connection**:

- **create connection:**

```
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 hide_maintuser_pwd to '{on | off}'
...
go
```

- **alter connection:**

```
alter connection to data_server.database
set hide_maintuser_pwd to '{on | off}'
go
```

- Server level – use **configure replication server**:

```
configure replication server
set hide_maintuser_pwd to '{on | off}'
go
```

The default is off.

If multiple connections to the same or a different database controlled by Replication Server share the same maintenance user, Replication Server alters the maintenance user password for all the connections if you set `hide_maintuser_pwd` on.

Maintenance User Password Status

Use the `rs_helpuser` stored procedure to display the password status of the maintenance user.

Sample output for `rs_helpuser` for TOKYO_RS Replication Server:

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 subscri

Maintenance Users		
User name	Destination DS.DB	Password Status
TOKYO_RSSD_maint	TOKYO_DS.TOKYO_RSSD	Initial
pubs2_maint	TOKYO_DS.pubs2	Hidden
pubs2_maint	SYDNEY_DS.pubs2sb	Hidden

Maintenance User Password Expiration Interval

Use **maintuser_pwd_expiration** to set the password expiration interval for the maintenance user at the server level.

Replication Server automatically changes the password when the password expires. Before you set a nonzero value for **maintuser_pwd_expiration**, you must enable maintenance user password protection by setting **hide_maintuser_pwd** to on.

For example, to set **maintuser_pwd_expiration** to 14 days, enter:

```
configure replication server
set maintuser_pwd_expiration to 14
```

The range of values is 0 to 32,767 days. The default password expiration interval for the maintenance user is the value set in the Replication Server **password_expiration** option. The default for **password_expiration** is 0 days which means the password does not expire.

Guidelines and Limitations for Maintenance User Security Configuration

There are several considerations for maintenance user security configuration.

- Multiple connections – if there are connections from different databases in the same Adaptive Server to a Replication Server and these connections share the same Adaptive Server login for the maintenance user, turning on **hide_maintuser_pwd** for one connection affects all connections. Replication Server synchronizes any change of password across all the connections.

However, if there are connections from databases controlled by different Replication Servers, turning on **hide_maintuser_pwd** for one connection does not affect the other connections as Replication Server does not support the synchronization of maintenance user passwords across multiple Replication Servers.

- Master database replication – Replication Server does not replicate the change in the maintenance user password when you enable maintenance user security with **hide_maintuser_pwd**. Each connection between a database and a Replication Server separately manages the random password change for the connection resulting from turning on **hide_maintuser_pwd**.
- Database restrictions – you cannot use **hide_maintuser_pwd** for non-Adaptive Server databases or the Replication Server ERSSD or RSSD.
- External authentication methods – you cannot set **hide_maintuser_pwd** on if the Adaptive Server database uses external authentication methods such as LDAP, KERBEROS, or PAM for the maintenance user. Instead you must use ASE authentication.

To use ASE authentication for the maintenance user, or verify that ASE authentication is in use, execute the Adaptive Server **alter login profile** command and the **modify authenticate**

with **ASE** clause and specify the maintenance user. If the command is successful, you can configure **hide_maintuser_pwd** for the maintenance user.

See **alter login** in the *Adaptive Server Enterprise Reference Manual: Commands* and *Changing Login Accounts* in the *Adaptive Server Enterprise Security Administration Guide*.

Disabling Maintenance User Password Protection

Disable maintenance user password protection for the connection to a replicate Adaptive Server database.

1. At the Replication Server, set **hide_maintuser_pwd** off for the connection to the replicate Adaptive Server database:

```
alter connection to data_server.database
set hide_maintuser_pwd to 'off'
go
```

2. Log in with the **sso_role** privilege to **isql** at the Adaptive Server replicate database and disable replication:

```
set replication off
go
```

3. At the Adaptive Server replicate database change the maintenance user password to one that is known to the administrator:

```
alter login maintenance user ID
modify password new_maintenance_user_password
go
```

4. Log in to Replication Server and change the password for all connections affected by the change to the maintenance user password to be the same as the password at the database:

```
alter connection to data_server.database
set password [to] new_maintenance_user_password
```

5. Enable replication. At the replicate Adaptive Server database, enter:

```
set replication on
go
```

Recovering from RSSD or Replicate Database Failure

If **hide_maintuser_pwd** is on and the RSSD, ERSSD or replicate Adaptive Server database fails, you may need to reset the maintenance user password after you use a backup to recover from the failure.

Prerequisites

Recover from the RSSD or ERSSD failure. See *Recovery from RSSD Failure* in the *Replication Server Administration Guide Volume 2*

Task

If you enable maintenance user password protection, the maintenance user password that was encrypted and stored in the RSSD that you restored from an earlier backup may not match the

current encrypted password in the Adaptive Server replicate database. Similarly, the maintenance user password in the replicate Adaptive Server database that you restored may not match the current password in the RSSD. After a restore, you must reset the password before you reenable maintenance user password protection.

1. Disable maintenance user password protection for all database connections.

```
configure replication server
set hide_maintuser_pwd off
```

2. Log in with the `sso_role` privilege to `isql` at the Adaptive Server replicate database and disable replication:

```
set replication off
go
```

3. At the Adaptive Server replicate database change the maintenance user password to one that is known to the administrator:

```
alter login maintenance user ID
...
modify password new_maintenance_user_password
...
go
```

4. Log in to Replication Server and change the password for all connections affected by the change to the maintenance user password to be the same as the password at the database:

```
alter connection to data_server.database
set password [to] new_maintenance_user_password
```

5. Enable replication. At the primary Adaptive Server database, enter:

```
set replication on
go
```

6. Enable maintenance user security for all database connections.

```
configure replication server
set hide_maintuser_pwd on
```

See also

- *Manage the RSSD* on page 96
- *ERSSD Recovery Procedures* on page 102

Password Encryption

Replication Server encrypts all passwords, and stores and transmits passwords in encrypted format.

Replication Server uses password encryption instead of clear text when storing all passwords for new Replication Server installations.

When you specify or change passwords with `rs_init`, `create user`, `alter user`, `create connection`, `alter connection`, `create route`, and `alter route`, Replication Server uses an algorithm to encrypt all passwords in the `rs_users` and `rs_maintusers` RSSD system tables, and in the Replication Server configuration file. You cannot decrypt the passwords.

Manage Replication Server Security

Replication Server uses the `rs_password_key` row in the `rs_encryptionkeys` RSSD system table, and the **RS_random** attribute in the configuration file to support password encryption and decryption. Replication Server automatically generates installation specific random values for the `rs_password_key` row in the system table and for the **RS_random** attribute when you start Replication Server and Replication Server does not find the values in the table or the configuration file.

You can regenerate random values for the password encryption key in the system table and configuration file with the **alter encryption key password_key_row_name regenerate** command. To regenerate the password encryption key in the `rs_password_key` row of `rs_encryptionkeys`, enter:

```
alter encryption key rs_password_key regenerate
```

Warning! Do not change or delete the **RS_random** attribute in the configuration file manually as this prevents Replication Server from starting.

Replication Server cannot retrieve the encrypted passwords and cannot start if:

- You do not backup and restore the configuration file with the corresponding RSSD or ERSSD.
- The **RS_random** attribute is lost or corrupted

If all users cannot log in because there is no valid **RS_random** attribute, remove the **RS_random** attribute in the configuration file if the attribute exists, and reset the sa user password. You can then log in to Replication Server and manually set the passwords for all users and maintenance users.

For upgrade and downgrade considerations, see *Replication Server Configuration Guide > Password Encryption*.

See also

- *Resetting a Lost or Forgotten sa User Password* on page 250

Send Encrypted Passwords for Replication Server Client Connections

Replication Server supports the **-X** option in **isql** that sends encrypted passwords through the network when making a client connection

To ensure that all Replication Server client connections—except the first connection to the RSSD—send encrypted passwords, set the Replication Server configuration parameter **send_enc_password** to “on.” For example, enter:

```
configure replication server
set send_enc_password to 'on'
```

To ensure that all Replication Server client connections, including the first connection to the RSSD, send encrypted passwords, set the configuration parameter **RS_send_enc_pw** to “on” in the `rs_name.cfg` file using a text editor.

If `RS_send_enc_pw` is “on,” all Replication Server connections to the RSSD send encrypted passwords—even if `send_enc_password` is “off.”

Existing Encrypted Password Migration

Newly created passwords use the Federal Information Processing Standards (FIPS)-certified 140-2 encryption algorithm.

Use the commands in the table to migrate existing encrypted passwords in the Replication Server configuration file, and in the `rs_users` and `rs_maintusers` tables.

Table 14. Commands to Encrypt Passwords in New Algorithm

Task	Command/Step
Encrypt existing user passwords to the new algorithm	<pre>alter user <i>user</i> set password <i>password</i></pre> <p>where:</p> <ul style="list-style-type: none"> <i>user</i> is the login name of the existing user <i>password</i> is the existing password you want to encrypt using the new algorithm.
Encrypt existing database maintenance user passwords to the new algorithm	<pre>alter connection to <i>data_server.database</i> set password to <i>password</i></pre> <p>where, <i>password</i> is the existing password you want to encrypt using the new algorithm.</p>
Encrypt existing route user passwords to the new algorithm	<pre>alter route to <i>dest_replication_server</i> set password to <i>passwd</i></pre> <p>where:</p> <ul style="list-style-type: none"> <i>dest_replication_server</i> is the name of the destination Replication Server <i>passwd</i> is the existing password you want to encrypt using the new algorithm.
Encrypt existing user passwords in the configuration file to the new algorithm	<ul style="list-style-type: none"> Use <code>rs_init</code> to encrypt the passwords using the new algorithm.

Extended Password Encryption Support

With version 15.1 and later, Replication Server uses Sybase Common Security Infrastructure (CSI) to provide server or client authentication, provide key-pair generation to support extended password encryption, and provide cryptography for encryption and decryption of passwords that are transmitted in the network between Replication Servers, and between Replication Server and the primary and replicate data servers.

Extended password encryption uses asymmetric key encryption, which allows Open Client applications with connection property `CS_SEC_EXTENDED_ENCRYPTION` enabled to

Manage Replication Server Security

connect to the Replication Server. It also allows Replication Server to enable **CS_SEC_EXTENDED_ENCRYPTION** when connecting to other servers.

Asymmetric key encryption uses a public key to encrypt the password and a private key to decrypt the password. The private key is not shared across the network, and is therefore secure.

Note: To use the extended password encryption feature, you must have a server that supports extended password encryption, such as ASE 15.0.2 ESD #2 or later.

Sybase Central Dependencies

The Replication Manager logs into the Replication Server and the RSSD using the login names and passwords that you specified when you added the server to Replication Manager.

If you are using Replication Manager, be sure to update the login information. This information can be found in the Replication Server properties dialog.

Replication Server Object Creation Dependencies

Login name and password dependencies also apply when you create Replication Server objects, specifically subscriptions and replicated functions (applied and request functions) that are executed at primary or replicate Replication Servers.

Subscriptions

When you create a subscription, the login name that you used to log in to the replicate Replication Server must exist on both the primary Replication Server and the primary Adaptive Server. The login name must have the same password on all three servers.

When you drop a subscription, the replicate Replication Server logs in to the primary Replication Server using the login name and password you used to log in to the replicate Replication Server. Do not change the password of this login name on the primary Replication Server before the **drop subscription** process is complete.

The RSSD “primary” user login name that is automatically created on the Replication Server is used as the “subscribing user” when routes are created. The rules for a user creating a subscription apply to the RSSD primary user.

Suggestions for subscriptions:

- Do not create subscriptions as the **sa** user.
- The **select** command, issued at the primary Replication Server when creating the subscription, does not include a table owner name unless an owner name is specified in the replication definition. If no owner name is specified, make sure that either the user owns the table or the table is owned by the “dbo” user.
- Do not change passwords while subscriptions are materializing or dematerializing.

Replicated Functions and Stored Procedures

When a primary Replication Server receives a request function or a request stored procedure from a replicate Replication Server, it logs in to the primary data server with the login name

and password of the user who initiated the request function or request stored procedure at the replicate site.

Therefore, to execute a request function or request stored procedure at a replicate data server, the user must have the same login name and password at the primary data server, and must have **execute** permission for the stored procedure at the primary data server.

When a replicate Replication Server receives an applied function or applied stored procedure from a primary site, the replicate Replication Server uses the maintenance user login name and password to execute the stored procedure in the replicate database.

Manage Replication Server User Security

Replication Server has its own set of login names, which are separate from data server login names. Replication Server login names are required so that administrative users of the system can execute Replication Server commands.

Users do not need Replication Server login accounts to access data replicated by Replication Server. Replicated data is available to users if they have permissions to access specific databases. The Database Administrator is responsible for creating databases and authorizing access to them.

Password encryption for users can be enabled or disabled.

See also

- *Examine Users, Passwords, and Permissions* on page 257

Manage Replication Server Login Names and Passwords

The replication system administrator, or any other user who has **sa** permission, can manage login names, and implement and administer password security.

Table 15. Commands for Managing Login Names

Command	Task
create user	Create a new login name.
alter user	Change the password for a login name.
drop user	Drop a user login name.

Concealing Password Input

Use the **isql --conceal** option, with the ":?" wildcard characters, to conceal the passwords you set when you create or alter users.

1. Start **isql** with the **--conceal** option before you create a user or change a password for a user.

Manage Replication Server Security

For example, at the `ny_rs` Replication Server for the `test_2` user, enter:

```
isql -Sny_rs -Utest_2 --conceal
Password:
```

2. Enter the **alter user** or **create user** command, and instead of typing a password in clear text, enter the `:?` wildcard pair of characters at the beginning of the command line for password entry.

For example, to change the password for the `test_2` user:

```
alter user test_2
set password
:?
verify password
:?
go
```

Optionally, use the double wildcard character pair `":?:"` within the same line to prompt for confirmation of the new password. For example:

```
alter user test_2
set password
:?:?
verify password
:?
go
```

isql prompts you for the information to replace the wildcard pair of characters. Each `:?` wildcard character pair you enter results in a separate `:?` prompt. Replication Server does display any characters you type at the `:?` prompt.

3. Enter the new password at the first `:?` prompt, which corresponds to the **set password** clause, and enter the old password at the second `:?` prompt, which corresponds to the **verify password** clause.

```
:? new_password
:? old_password
```

If you use the double `":?:"` character pairs option in the command, you see:

```
:?
Confirm :?
:?
```

Enter the new password at the first `:?` prompt corresponding to the **set password** clause, enter the new password again at the second `Confirm :?` prompt, and enter the old password at the third `:?` prompt corresponding to the **verify password** clause.

If the password you enter complies with the password security requirements, you see:

```
User 'test_2' is altered.
```

Create a Replication Server Login Name

Use the **create user** command to add new user login names to Replication Server.

You must specify a password for the user when the login name is created. If the user has no password, you must set the password to “null,” which specifies an empty string.

The **create user** command requires **sa** permission. The syntax for **create user** is:

```
create user user
set password {new_password | null}
[set password_parameter to 'parameter_value']
```

A user's password can be up to 30 characters long and include letters, digits, and symbols. Case is significant. If the password contains spaces, enclose the password in single quotation marks.

Users can change their own passwords using the **alter user** command.

To create a login name for the user “thomk” with the password “vacUUm”, enter:

```
create user thomk
set password vacUUm
```

As the administrator, you can set a password expiration interval for a user. For example, to create a user named jsmith having an initial password of 1Buiopr89, with a password expiration interval of 90 days, enter:

```
create user jsmith
set password to 1Buiopr89
set password_expiration to '90'
```

See *Replication Server Reference Manual* > *Replication Server Commands* > **create user**.

See also

- *Permission Summary* on page 253

Change a Replication Server Password

The replication system administrator can change the password of any user with the **alter user** command. A user can change his or her own password.

The syntax for **alter user** is:

```
alter user user
set password {new_password | null}
[verify password old_password]
[set password_parameter to 'parameter_value']
```

The same rules that you use for specifying the password using **create user** also apply to **alter user**.

The **verify password** clause, which prevents users from altering each other's passwords, is required for users without **sa** permission.

For example, if the user with login name “louise” wants to change her own password from “EnnuI” to “somNIfic”, she should enter:

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

Manage Replication Server Security

As the administrator, you can set the password expiration interval for the user. For example, to change the password and password expiration interval of user jsmith to 60 days, enter:

```
alter user jsmith
set password to newpass
set password_expiration to '60'
```

See *Replication Server Reference Manual > Replication Server Commands > alter user*.

Password Configuration Options for All Users

There are several **configure replication server** options you can use to implement and administer Replication Server password security for all users. Replication Server uses password encryption instead of clear text when storing all passwords for new Replication Server installations.

Syntax

```
configure replication server
set password_param to 'parameter_value'
```

Password Configuration Options

password_parameter	Description
min_password_len	Minimum number of characters required. <ul style="list-style-type: none">• 0 – no minimum length.• Range – 6 to 16 (default 6).
max_password_len	Maximum number of characters. Always set max_password_len to a value greater than min_password_len . Range – 13 to 30 (default 30).
password_lowercase_required	Whether lowercase characters are required. <ul style="list-style-type: none">• True – required.• False – not required (default).
password_uppercase_required	Whether uppercase characters are required. <ul style="list-style-type: none">• True – required.• False – not required (default).
password_numeric_required	Whether a numeric character is required. <ul style="list-style-type: none">• True – required.• False – not required (default).

<i>password_parameter</i>	Description
password_special_required	<p>Whether a special character is required.</p> <ul style="list-style-type: none"> • True – required. • False - not required (default).
simple_passwords_allowed	<p>If you set this option (or "simple_passwords_allowed") to false, Replication Server does not allow the password to contain the user name or any values from a user password dictionary.</p> <ul style="list-style-type: none"> • True – allowed (default). • False – not allowed. <p>You can create the password dictionary in the RSSD in the <code>rs_dictionary</code> system table. The table does not store default values. You must create your own scripts to insert values into the table. For example:</p> <pre>insert into rs_dictionary (words) values ("abcd"); insert into rs_dictionary (words) values ("1234");</pre>
disallowed_prev_passwords	<p>Number of previous passwords that cannot be reused when the user changes his or her password.</p> <ul style="list-style-type: none"> • 0 – previous passwords allowed. • Range – 0 to 32,767 (default 0). <p>The parameter value does not apply to a user password when the administrator is resetting the password.</p>

password_parameter	Description
<p>password_expiration</p>	<p>Number of days after which the password expires.</p> <ul style="list-style-type: none"> • 0 – password never expires (default). • Range – 0 to 32,767. <p>You can use password_expiration with alter user and create user.</p> <p>If the password has expired, Replication Server locks the account and notifies the user that the password has expired. If the user does not reset his password, he or she cannot log in once disconnected until the administrator resets the password. The new password must meet all the password requirements.</p> <p>Passwords do not expire for any user that rs_init creates with connect source permission or the ID user. These passwords override any setting for password_expiration that you set for all users in the Replication Server. Databases, other Replication Servers, and Replication Agents use user IDs with connect source permission.</p> <p>Administrators set the password to not expire for any user that is created for Replication Agent or an RSI.</p>
<p>initial_password_expiration</p>	<p>Number of days after which the initial password expires.</p> <ul style="list-style-type: none"> • 0 – initial password never expires. • Range – 0 to 32,767 (default 0). <p>An initial password for a user is the password set by the administrator when creating the user or when resetting the user password.</p>

<i>password_parameter</i>	Description
max_failed_logins	<p>Maximum number of failed login attempts Replication Server allows before locking the account.</p> <ul style="list-style-type: none"> 0 – account never locked. Range – 0 to 32,767 (default 0). <p>Replication Server locks the account according to the time interval set in password_lock_interval.</p>
password_lock_interval	<p>Number of minutes that an account remains locked if the user reaches the maximum number of login attempts set in max_failed_logins.</p> <ul style="list-style-type: none"> 0 – account remains locked until administrator resets password. Range – 0 to 32,767 (default 0).
unused_login_expiration	<p>Number of days after which an unused user account expires.</p> <ul style="list-style-type: none"> 0 – unused account never expires. Range – 0 to 32,767 (default). <p>Replication Server locks an account that remains unused for longer than unused_login_expiration. The administrator can reactivate the account by resetting the password.</p>

Examples

- Enforce the minimum password length as 8 characters for all users:

```
configure replication server
set min_password_len to '8'
```

- Enforce the password expiration interval as 90 days for all users:

```
configure replication server
set password_expiration to '90'
```

Usage

- password_expiration** is the only parameter you can use with **alter user** and **create user**. See **alter user** and **create user** in *Replication Server Reference Manual > Replication Server Commands*.
- Password settings that you set for individual users with **create user** or **alter user** override any value that you set with **configure replication server**.
- Use the **rs_helpuser** stored procedure to display the status and settings of current users.

Manage Replication Server Security

- Minimum password length:
 - When the administrator or the user changes a user password, the new one must comply with any system minimum length that has been set with **configure replication server**.
 - Setting the minimum password length does not impact the current passwords of users until users change the password.
- Password expiration interval:
 - When the administrator or the user changes the password of the user, Replication Server records the date. When the user logs in, Replication Server checks the login date against the password expiration setting. If there is a password expiration value set either for the user or at the system level, and Replication Server determines that the password has expired, Replication Server notifies the user to change the password and locks the account. Replication Server unlocks the account only when the user enters a new password that meets requirements. If the user disconnects before changing the password, the administrator must reset the password.
 - Sybase recommends that you set the **password_expiration** to 0 for users that have "connect source" privilege, such as Replication Agent users, to prevent their passwords from expiring and interfering with replicating data.
- When the LDAP user authentication is enabled, the password applies to both Replication Server and LDAP user accounts. Ensure that you enter a valid password that adheres to Replication Server password policy.
- The **password_encryption** parameter is deprecated.

Permissions

You must have sa permission to configure password parameters.

Drop a Replication Server Login Name

Use the **drop user** command to remove an existing login name from the Replication Server.

drop user requires **sa** permission. The syntax for **drop user** is:

```
drop user user
```

For example, the following command removes the "thomk" login name:

```
drop user thomk
```

Resetting a Lost or Forgotten sa User Password

Use the **reset_password=sa** parameter to reset the password for the sa user if you lose or forget the password. You cannot use the parameter to reset passwords of any other account.

1. Add the **reset_password=sa** parameter to the Replication Server configuration file.

Note: You cannot use **isql** to run **reset_password=sa**.

2. Restart Replication Server.

During start-up, Replication Server generates a random password for one-time use for the sa user. The **reset_password=sa** parameter is then removed from the configuration file to prevent password from being generated during subsequent restarts.

3. Log in as the sa user using the generated one-time password.

The password expires immediately, and Replication Server locks the sa user account to prevent another user from attempting to log in using the same password. Replication Server notifies the sa user that the password has expired.

4. Enter a new password that meets all the requirements.

Manage Replication Server Permissions

Replication system administrators manage Replication Server permissions with the **grant** and **revoke** commands. Permissions determine which RCL commands users are permitted to execute.

Any user with a Replication Server login name can execute all **admin** commands and the **check subscription** command. Other commands can be executed only by users who have been granted the required permissions.

Replication Server Permissions

Replication Server users can be granted any of four permissions.

Table 16. Replication Server Permissions

Permission	Description
sa	Users with sa permission are Replication system administrators. They can execute any Replication Server command and may grant and revoke other permissions, including sa , to and from other users.
create object	Users with create object permission can create objects such as replication definitions, subscriptions, and function strings. Users with create object permission automatically have primary subscribe permission.
primary subscribe	Users with primary subscribe permission can execute the commands needed to create subscriptions for primary data stored in databases managed by the Replication Server. Users with primary subscribe permission at the primary site and create object permission at the replicate site can create a subscription for data at the primary site, but cannot create replication definitions or function strings at the primary site.

Permission	Description
connect source	<p>The connect source permission is required for:</p> <ul style="list-style-type: none"> • Login names that RepAgents use to log in to Replication Server, allowing RepAgent to execute the subset of RCL commands known as Log Transfer Language (LTL). • Login names that a source Replication Server uses to connect to a destination Replication Server for the purpose of sending replicated data or replicated functions. You provide this login name using the create route command.

Requirements for Creating Subscriptions

Learn what permissions you require before you can create subscriptions.

A subscription creator must have accounts on both the primary and replicate Replication Servers, and the accounts must have the same login name and password. The subscription creator enters a command or a series of commands at the replicate Replication Server, which passes the request to the primary Replication Server.

When the optional clauses **use dump marker** and **subscribe to truncate table** are used, the login name and password for the replicate Replication Server should be the same for the primary Replication Server, as well as for both the primary and replicate databases.

At the replicate Replication Server (the destination of the subscription data), the subscription creator must have, at minimum, **create object** permission in order to materialize the subscription.

At the primary Replication Server (the source of the subscription data), the subscription creator must have, at minimum, **primary subscribe** permission in order to enter at the replicate site all commands involved in creating subscriptions:

- **create subscription** (for atomic and nonatomic materialization)
- **define subscription** (for bulk materialization)
- **activate subscription** (for bulk materialization)
- **validate subscription** (for bulk materialization)
- **drop subscription**

The **primary subscribe** permission, a subset of **create object** permission, is provided at the primary Replication Server. It lets users at replicate sites create subscriptions to data stored at primary sites. From replicate sites, these users cannot create any other objects at primary sites, only subscriptions.

Note: Users with **create object** and **sa** permissions can also create subscriptions from replicate Replication Servers. The minimal permission required at the primary Replication Server for a user at a replicate site to create subscriptions is **primary subscribe**.

A user creating a subscription must have the following Adaptive Server permissions:

- **select** permission on the table in the primary database
- **insert**, **update**, and **delete** permission on the replicate table
- **execute** permission on the **rs_marker** stored procedure in the primary database

If you are a replication system administrator, restrict **primary subscribe** and **create object** permissions at primary sites to users who require them in order to create subscriptions.

It is possible for a user who has **primary subscribe** or **create object** permission to begin creating a subscription without having **select** permission on the table. If this occurs, Replication Server responds in the following manner:

- If the subscription is created with atomic materialization, the **select with holdlock** operation fails at the primary database during materialization. The subscription retry daemon (dSUB) retries the **select with holdlock** until the subscription is dropped or until the **select** permission is granted to the user for the table at the primary database.
- If the subscription is created with nonatomic materialization, the **select** operation fails at the primary database during materialization. The subscription retry daemon (dSUB) retries the **select** until the subscription is dropped or the **select** permission is granted.
- If the subscription is created with bulk materialization, there is no **select** transaction, so no error messages are logged, and the subscription succeeds.

Permission Summary

You must have the minimum permission required to execute each RCL command.

If you have **create object** permission, you automatically have **primary subscribe** permission. If you have **sa** permission, you can execute any command.

Table 17. Minimum Permissions to Execute RCL Commands

To Execute	Minimum Permission Required
abort switch	sa
activate subscription	create object at replicate, primary subscribe at primary
create partition	sa
admin commands	Can be executed by any user
allow connections	sa
alter connection	sa
alter database replication definition	create object
alter function	create object
alter function replication definition	create object
alter applied function replication definition	create object

To Execute	Minimum Permission Required
alter request function replication definition	create object
alter function string	create object
alter function string class	sa
alter logical connection	sa
alter partition	sa
alter queue	sa
alter replication definition	create object
alter route	sa
alter user	sa – Users can change their own passwords by including the verify clause
assign action	sa
check publication	Can be executed by any user
check subscription	Can be executed by any user
configure connection	sa
configure logical connection	sa
configure replication server	sa
configure route	sa
create article	create object
create connection	sa
create database replication definition	create object
create error class	sa
create function	create object
create function replication definition	create object
create applied function replication definition	create object
create request function replication definition	create object
create function string	create object

To Execute	Minimum Permission Required
create function string class	sa
create logical connection	sa
create partition	sa
create publication	create object
create replication definition	create object
create route	sa
create subscription	create object at replicate, primary subscribe at primary
create user	sa
define subscription	create object at replicate, primary subscribe at primary
drop article	create object
drop connection	sa
drop database replication definition	create object
drop error class	sa
drop function	create object
drop function replication definition	create object
drop function string	create object
drop function string class	sa
drop logical connection	sa
drop partition	sa
drop publication	create object
drop replication definition	create object
drop route	sa
drop subscription	create object at replicate, primary subscribe at primary
drop user	sa
grant	sa
ignore loss	sa
move primary	sa
rebuild queues	sa

To Execute	Minimum Permission Required
resume connection	sa
resume distributor	sa
resume log transfer	sa
resume queue	sa
resume route	sa
revoke	sa
set proxy	sa
set autocorrection	create object
set log recovery	sa
shutdown	sa
suspend connection	sa
suspend distributor	sa
suspend log transfer	sa
suspend route	sa
switch active	sa
sysadmin commands	sa
validate subscription	create object at replicate, primary subscribe at primary
wait for create standby	sa
wait for switch	sa

Grant Permissions

Use the **grant** command to grant permissions to users.

The ability to grant and revoke permissions is reserved for the replication system administrator. Any user who has been granted **sa** permission can play the role of replication system administrator, and can transfer the **grant** and **revoke** ability to other users by granting them **sa** permission.

The syntax for the **grant** command is:

```
grant
{sa | create object | primary subscribe | connect source}
to user
```

The *user* is the login name of the user to receive the permission. You can grant only one permission at a time.

Permissions are assigned to Replication Server users—not to database users. A Replication Server user who has **create object** permission can create Replication Server objects that are associated with any database managed by the Replication Server.

In the following example, the replication system administrator grants **create object** permission to the “thomk” login name:

```
grant create object to thomk
```

Revoke Permissions

Use Sybase Central or the **revoke** command to remove permissions previously granted to a user.

The syntax for the **revoke** command is:

```
revoke {sa | create object | primary subscribe |
connect source}
from user
```

Note: You cannot revoke the **sa** permission from, or drop, the **sa** login name. This ensures that Replication Server is never without a replication system administrator.

The four permissions are managed independently. They can be granted and revoked in any order and the result is the same.

The following **revoke** command prevents user “louise,” who does not have **sa** permission, from creating replication definitions:

```
revoke create object from louise
```

Examine Users, Passwords, and Permissions

You can display the login names, passwords, and permissions for Replication Server users and threads by using the **rs_helpuser** stored procedure or by querying the **rs_maintusers** and **rs_users** system tables in the RSSD. You can also use Sybase Central to view information on Replication Server login names.

Use the rs_helpuser Stored Procedure

Use the **rs_helpuser** stored procedure to display information about user login names known to a Replication Server.

The syntax for **rs_helpuser** is:

```
rs_helpuser [user]
```

With no parameters, **rs_helpuser** displays information about all user login names known to the current Replication Server. Permissions are displayed for each primary or maintenance user login name.

If you supply a login name parameter, **rs_helpuser** displays information about that login name only.

Query the rs_maintusers System Table

The `rs_maintusers` system table in the RSSD contains the login name and password information for maintenance users.

`rs_maintusers` includes a column to identify if the password is encrypted or clear text, and a column to hold the encrypted password.

For example, the following query, executed in the RSSD, lists all available information, including login names, for maintenance users:

```
select * from rs_maintusers
```

Query the rs_users System Table

The `rs_users` system table in the RSSD contains the login name and password information for Replication Server users.

`rs_users` also includes a column to identify if the password is encrypted or clear text, and a column to hold the encrypted password.

The `rs_users` system table also includes a `permissions` column, which stores the permissions for each login name. The `permissions` column is a bit-mask of the permissions granted to users.

For example, the following query, executed in the RSSD, lists users who have **sa** permission:

```
select username, uid from rs_users
where permissions & 0x0001 != 0
```

Permission Bitmask Values in the rs_users System Table

Each type of permission has a bitmask value.

Table 18. Permission Bitmask Values in the `rs_users` System Table

Permission	Mask Value
sa	0x0001
connect source	0x0002
create object	0x0004
primary subscribe	0x0008

Replication Server Gateway

With version 15.2, Replication Server introduces Replication Server gateway, which minimizes explicit login into the different servers, and makes managing a replication system easier.

In managing a replication system, the replication system administrator (RSA) must log in to multiple replication servers, ID servers, and the corresponding Replication Server System Database (RSSD). The RSA must also frequently switch logins between Replication Server and the RSSD.

The Replication Server gateway uses your RSSD primary user name and password to log in to RSSD, your ID server user name and password to log into ID Server, your remote server identification (RSI) to log into a remote Replication Server, and your maintenance user ID to log into the remote Adaptive Server. You do not need to supply this information more than once, when you access Replication Server itself.

Cascading Connection

The Replication Server gateway also supports cascading connections, which allow your Replication Server to communicate with servers that it is not directly connected to.

It also allows you to manage a replication domain using a single client connection. For example, you can connect to an ID Server, and then to the ID Server's RSSD. In this case, both the primary, controlling Replication Server and the ID server are gateways; commands pass through to the ID server's RSSD, and result sets pass back through to you.

Enable the Replication Server Gateway

Use the **connect** command to turn Replication Server into a gateway to its RSSD, ID server, or to a remote Replication Server.

Note: Issuing the **connect** command requires an **sa** role for the first login to Replication Server.

Syntax:

```
connect [to] [rssd | idserver | srv_name | ds_name.db_name]
```

Parameters:

- **rssd** – turns Replication Server into a gateway to its RSSD. Allows the gateway to use *RSSD_primary_user* and *RSSD_primary_pw* entries in its configuration file.
- **idserver** – turns Replication Server into a gateway to its ID server, providing that the Replication Server itself is not the ID server. Allows the gateway to use *ID_user* and *ID_pw* entries in the configuration file.

Manage Replication Server Security

- *srv_name* – the name of the remote Replication Server you want the gateway to connect to. The Replication Server gateway uses RSI to log into the remote server, and requires a direct route to the remote server.

Note: Replication Server cannot directly connect to itself. However, you can work around this by using a cascading connection.

- *ds_name.db_name* – the name of the remote data server and database that you want the gateway to connect to. The Replication Server gateway uses the maintenance user to log into the remote data server. This allows you to perform tasks that maintenance users of the designated database are permitted to do. However, you cannot access the other databases defined in the data server you connected to.

Replication Server gateway can directly connect to Adaptive Server and Sybase® IQ as well as replicate data servers that are supported by Enterprise Connect Data Access (ECDA). You cannot use Replication Server gateway to connect to replicate data servers if you are using one of the ExpressConnect products to communicate with those servers.

Track Connections

Use **show connection** and **show server** to manage your cascaded connection.

Cascaded connections created in the gateway are kept in a connection stack, with the Replication Server that issued the first **connect** command placed at the bottom of the stack.

- **show connection** – lists the contents of the connection stack.
- **show server** – displays the current working server.

Drop Connections

Use **disconnect** command to terminate a connection to a server.

disconnect exits the connection stack one at a time. To exit from all the connections, use **disconnect all**.

```
{disconnect | disc} [all]
```

The **disconnect** command behaves differently in Replication Server 15.1 and earlier.

In Replication Server 15.1 and earlier, a **disconnect** command terminates the gateway mode, and returns the working server status to the Replication Server that issued the first **connect** command. When your connection stack includes Replication Server versions 15.2, and 15.1 or earlier, and you issue a **disconnect** command, the **show connection** and **show server** commands may not display the expected output.

For details about the commands, see *Replication Server Reference Manual*.

Limitations to Replication Server Gateway

When using Replication Server gateway, the client and the server must use the same locale set because Replication Server cannot perform character set conversion. You cannot use

Replication Server gateway to connect to non-Adaptive Server data servers when Replication Server is using ExpressConnect to communicate with those servers.

Manage Network-based Security

Learn about Replication Server support for third-party, network-based security mechanisms.

In a client/server environment, it is important to provide secure data pathways so data transmission remains confidential. Replication Server supports third-party, network-based security mechanisms that focus on:

- Authentication and unified login
- Secure message transmission

With network-based security, users are authenticated—the process of verifying that users are who they say they are—by the security system at login. They receive a credential that can be presented to remote servers in lieu of a password. As a result, users have seamless access to the components of the replication system through a single login.

Replication Server version 12 and later supports MIT Kerberos version 5 or later, CyberSafe Kerberos version 5 Security Server, and Transarc DCE version 1.1 Security Server.

Depending on which of these security mechanisms you choose, you can select one or more of these features to secure data transmission:

- Unified login – enables the user to log in to components of the replication system with a single credential issued by the security mechanism.
- Confidentiality – enables the sending and receiving of encrypted data.
- Integrity – ensures that data has not been tampered with.
- Replay detection – verifies that data has not been intercepted.
- Origin check – verifies the source of each data packet.
- Out-of-sequence detection – checks that data packets are received in the order sent.

The security mechanism allows Replication Server to establish secure connections with other Replication Servers, with Adaptive Server, and with other data servers that support the Kerberos or DCE security mechanism and certain Replication Server requirements. You choose the method or methods to secure data transmission between them.

How Security Services Work

How security services work depends on whether Replication Server (or Adaptive Server or other data server) is acting as client or server.

Clients use the security mechanism to ensure a secure pathway to a remote server. Replication Server logs in to remote servers (acting as a client) and also accepts incoming logins (acting as a server).

Replication Server, when acting as the client, uses the security mechanism to ensure a secure pathway to a remote Replication Server or Adaptive Server. Once the secure pathway is

Manage Replication Server Security

established, the security mechanism can provide message protection. When Replication Server acts as a server, it accepts or rejects logins based on its default security settings.

Login Authentication

Security services use login authentication.

If a client requests authentication services:

1. The client validates the login with the security mechanism and receives a credential, which contains relevant security information.
2. The client sends the credential to the server and informs the server it wants to establish a secure connection.
3. The server authenticates the client's credential with the security mechanism. If the credential is not valid, the secure connection is rejected.
4. The server checks message protection properties, if the properties are compatible, the connection is established.

Message Protection

Security services use message protection.

If the current Replication Server (the client) requests data protection services:

1. The client uses the security mechanism to prepare the data packet it will send to the server. For example, if the client requests message confidentiality, the security mechanism encrypts the commands that will be sent to the remote server. If the client requests out-of-sequence checking, the security mechanism time-stamps each data packet.
2. The client sends the data to the destination server.
3. When the server receives the data, it uses the security mechanism to perform the appropriate decryption or validation.
4. The server returns the results to the client, using the security mechanism to perform the security action requested. For example, the server returns results in encrypted form or time-stamps each data packet returned to the client.

Requirements and Restrictions

Learn the requirements and restrictions for network-based security.

To enable network-based security you need:

- A network-based security mechanism installed on all machines for which network security is to be enabled. The security mechanism must be supported by Replication Server.

Note: Make sure that you use either the MIT Kerberos, CyberSafe Kerberos or Transarc DCE security mechanism. Sybase network-based security will not run on other Kerberos or DCE security mechanisms.

- Replication Server version 11.5 or later for all client and destination Replication Servers.

- Adaptive Server version 12 or later and/or compatible heterogeneous data servers for all client and destination data servers. Compatible heterogeneous data servers must support the security mechanism installed on Replication Server and the set proxy concept. See the *Replication Server Heterogeneous Replication Guide*.

These restrictions apply:

- Both ends of a secured pathway (client and server) must support the same security mechanism, and the security parameters must have the same feature settings.
- User names must be unique throughout the replication system.

If your replication system supports multiple security systems, and you cannot guarantee unique user names, you may need to turn off request stored procedures to avoid a potential security breach.

See also

- *Maintain Network Security* on page 292
- *Potential Security Issue* on page 296

Set up Network-based Security

Network-based security requires several steps to set up.

1. Modify configuration parameters and environment variables, as necessary.
2. Identify the Replication Server principal user.
3. Activate the network-based security mechanism.
4. Start server and clients.
5. Configure security services for connections, routes, and other Replication Server pathways.

Modify Configuration Parameters and Environment Variables

Learn what are the configuration files you may need to configure for network security.

Configuration files are created during installation at default locations in the Sybase directory structure. The configuration files you may need to configure for network security are:

- `libtcl.cfg`
- `objectid.dat`
- The `interfaces` file

If you are using Kerberos security services, you may also need to modify the `KRB5_KTNAME` environment variable.

Configure libtcl.cfg

Drivers are libraries that provide an interface to an external service provider. Use the `libtcl.cfg` configuration file on a 32-bit machine or `libtcl164.cfg` configuration file

Manage Replication Server Security

on a 64-bit machine as a template into which you enter all the configuration information about the security drivers installed on a machine.

The `libtcl.cfg` file is located in the `$$SYBASE/$SYBASE_OCS/config/` directory in UNIX, or the `%SYBASE%\%SYBASE_OCS%\ini` directory in Windows. Refer to the *Open Client and Open Server Configuration Guide* for more information about Sybase drivers.

The syntax for a security driver entry is:

```
provider=driver init-string
```

where

- *provider* is the local name for the security mechanism, for example, “dce.” The mapping of the local name to a global object identifier is defined in `objectid.dat`.
 - The default local name for the DCE security mechanism is “dce.”
 - The default local name for the Kerberos security mechanism is “csfkrb5.”

If you use a local mechanism name other than the default, you must change the local name in the `objectid.dat` file.

- *driver* is the name of the security driver, for example, `libsdcce.so`.
- *init-string* is the initialization string for the driver.
 - For the DCE driver, use this syntax for *init-string*, where *cell_name* is the name of your DCE cell:

```
secbase=././cell_name
```

- For the Kerberos driver, use this syntax for *init-string*, where *domaine_name* is the name of your Kerberos domaine:

```
secbase=@domaine_name
```

Use a text editor to customize `libtcl.cfg` for your site. Make sure that lines you do not want are preceded with the “;” character. Change one parameter at a time and reboot Replication Server to effect the changes you make.

- An example of an entry for a DCE driver is:

```
[SECURITY]
dce=libsdcce.so secbase=././cell_name
```

- An example of an entry for a Kerberos driver is:

```
[SECURITY]
csfkrb5=libsybskrb.so
secbase=@ASE libgss=/krb5/lib/libgss.so
```

Configure objectid.dat

The `objectid.dat` file maps global object identifiers (OIDs) to local names.

It is located in the `$$SYBASE/config` directory in UNIX, or the `%SYBASE%\ini` directory in Windows. You may need to edit `objectid.dat` only if you have changed the local name of a security service in the `libtcl.cfg` file.

- A sample entry in the `objectid.dat` file for DCE is:

```
[secmech]
1.3.6.1.4.1.897.4.6.1 = dce, dcesecmech
```

- A sample entry in the `objectid.dat` file for Kerberos is:

```
[secmech]
1.3.6.1.4.1.897.4.6.6 = csfkrb5
```

Configure the interfaces File

The `interfaces` file contains network and security information for servers.

The `interfaces` file is located in `$SYBASE/interfaces` in UNIX, or `%SYBASE%\ini\sql.ini` in Windows. If you use network security, you must include a `secmech` line that gives the global identifiers of supported security services. Supported security mechanisms are listed by their OIDs. Multiple security mechanisms are separated by commas.

This is a sample entry for the `interfaces` file for either DCE or Kerberos where `server_principal_user_name` is the name of the Replication Server principal user:

```
server_principal_user_name
  query tcp ether plum 1050
  master tcp ether plum 1050
  secmech 1.3.6.1.4.1.897.4.6.6
```

See also

- *Identify the Principal User* on page 277

Set Environment Variables (Kerberos)

When Replication Server serves as a client, set the `KRB5CCNAME` to indicate the location of the credential cache.

The credential is a combination of a ticket-granting ticket and the ticket's session key. Replication Server uses this credential to identify itself when login to remote server.

If you are using the Kerberos network security, you may need to reset the `shared-library path` and `KRB5_KTNAME` environment variables.

- Make sure that the shared-library file is in a directory specified in the shared library path so that the client can find the shared-library file at runtime. Shared-library files are:
 - `libgss.so` on Sun Solaris
 - `libgss.sl` on HP-UX
- If the server key table file is in a location other than the Kerberos system default, set the `KRB5_KTNAME` environment variable to the fully qualified pathname of the key table file.
- If you want to use a principal name which is different from Replication Server name, set the `SYBASE_RS_PRINCIPAL` environment variable.

Manage Replication Server Security

- Make sure that the `LD_LIBRARY_PATH` environment variable includes the path to the CyberSafe/MIT `lib` directory as well as the `lib` directories for Adaptive Server, Open Client/Server, and Replication Server.
- Similarly, make sure that the `PATH` environment variable includes the path to the CyberSafe/MIT `bin` directory as well as the `bin` directories for Adaptive Server, Open Client/Server, and Replication Server.

Configuring the `rs_principal_users.cfg` File

If you modify the principal name of a Replication Server when multiple instances of Replication Servers are running, add the new principal name in the `rs_principal_users.cfg` configuration file,.

1. Open the `rs_principal_users.cfg` configuration file found in the `$SYBASE` folder.
2. Enter the principal name of a Replication Server that is modified and used for authentication with other Replication Servers.

For example, if there are ten Replication Servers with names and principal names as *PRS1*, *PRS1*, *PRS3*, and so on. If you modify the principal name of *PRS1* to *PRS1_principal*, enter the modified name in the `rs_principal_users.cfg` file as:

```
PRS1:PRS1_principal
```

If you modify the principal name of more than one Replication Server, enter one line for each server.

For example:

```
PRS2:PRS2_principal  
PRS3:PRS3_principal  
PRS4:PRS4_principal
```

Note: You can add the principal names of all Replication Servers in one `rs_principal_users.cfg` configuration file and copy this file to the same folder where the Replication Server configuration file is located; for example, `$SYBASE`.

3. Run **`sysadmin principal_users[,reload]`** to reload the principal names of all Replication Servers stored in the `rs_principal_users.cfg` configuration file.

See *Replication Server Reference Manual > Replication Server Commands > sysadmin principal_users[,reload]*.

Adaptive Server-to-Adaptive Server Replication with Kerberos Scenario

Set up Kerberos authentication for an Adaptive Server-to-Adaptive Server replication system.

Prerequisites

- Install Adaptive Server Enterprise 15.7 SP100 or later, and start the database. See the *Adaptive Server Installation Guide*.

- Install Replication Server 15.7.1 SP100 or later and start the server. See the *Replication Server Installation Guide*.
- Install the Kerberos key distribution center (KDC). To install the KDC for your Kerberos environment, see the vendor documentation.

Configuring Adaptive Server to Use Kerberos

Configure the Adaptive Server to use Kerberos security for data replication.

Prerequisites

Before you add the principal name in the KDC, ask the system administrator to provide the Kerberos library path.

Task

In this example, the primary Adaptive Server is *Kerberos_PDS* and its principal name is *Kerberos_PDS_principal*. The replicate Adaptive Server is *Kerberos_RDS* and its principal name is *Kerberos_RDS_principal*.

1. To generate the keytab file, and its credentials, add the Adaptive Server principal name in the KDC .

For example, on the MIT Kerberos machine, run:

```
export LD_LIBRARY_PATH=/mitkrb/lib:$LD_LIBRARY_PATH
```

```
export KRB5_CONFIG=/mitkrb/krb5.conf
```

```
export PATH=/mitkrb/linux/bin:$PATH
/mitkrb/linux/sbin/kadmin -p kadmin/admin -k -t /mitkrb/linux/
admin.key
```

```
kadmin:addprinc Kerberos_PDS_principal@ASE //password: a1234abcd
```

```
kadmin:ktadd -k /usr/Kerberos_PDS_principal.keytab
Kerberos_PDS_principal@ASE
```

```
kadmin:addprinc Kerberos_RDS_principal@ASE //password: a1234abcd
```

```
kadmin:ktadd -k /usr/Kerberos_RDS_principal.keytab
Kerberos_PDS_principal@ASE
```

```
kadmin:quit
```

```
export KRB5CCNAME=/usr/Kerberos_PDS_principal.cc
/mitkrb/linux/bin/kinit -f -k -t /usr/
Kerberos_PDS_principal.keytab Kerberos_PDS_principal@ASE
```

```
export KRB5CCNAME=/usr/Kerberos_RDS_principal.cc
/mitkrb/linux/bin/kinit -f -k -t /usr/
Kerberos_RDS_principal.keytab Kerberos_RDS_principal@ASE
```

where *Kerberos_PDS_principal* and *Kerberos_RDS_principal* are the principal names for the *Kerberos_PDS*, and the *Kerberos_RDS* respectively.

Manage Replication Server Security

2. Configure the `interfaces` file found in the `$SYBASE/interfaces` in UNIX, or `%SYBASE%\ini\sql.ini` in Windows.

Add this line:

```
secmech 1.3.6.1.4.1.897.4.6.6
```

```
Kerberos_PDS
  query tcp sun-ether replinuxb15 20001
  master tcp sun-ether replinuxb15 20001
  secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_PDS_principal
  query tcp sun-ether replinuxb15 20001
  master tcp sun-ether replinuxb15 20001
  secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_RDS
  query tcp sun-ether replinuxb15 20002
  master tcp sun-ether replinuxb15 20002
  secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_RDS_principal
  query tcp sun-ether replinuxb15 20002
  master tcp sun-ether replinuxb15 20002
  secmech 1.3.6.1.4.1.897.4.6.6
```

3. Configure the `libtcl.cfg` configuration file on a 32-bit machine or `libtcl64.cfg` configuration file on a 64-bit machine found in the `$SYBASE/$SYBASE_OCS/config/` directory in UNIX, or the `%SYBASE%/$SYBASE_OCS/ini` directory in Windows, by adding:

```
[SECURITY]
csfkrb5=libsybskrb64.so libgss=/krb5/lib/libgssapi_krb5.so
secbase=@ASE
```

or:

```
[SECURITY]
csfkrb5=libsybskrb.so libgss=/krb5/lib/libgssapi_krb5.so
secbase=@ASE
```

where:

- `csfkrb5` – is the default name for the Kerberos security mechanism.
- `driver` – is the name of the security driver, for example, `libgssapi_krb5.so`.
- `init-string` – is the initialization string for the driver, for example, `secbase=@ASE`.

Note: Ask the system administrator to provide the Kerberos library version and path.

4. If you use a local mechanism name other than the default name, configure the `objectid.dat` file.

The `objectid.dat` file is in the `$SYBASE/config` directory in UNIX, or the `%SYBASE%/ini` directory in Windows.

Add the local name in the `objectid.dat` file and ensure it includes this line:

```
"1.3.6.1.4.1.897.4.6.6 = csfkrb5"
```

5. To configure use security services, enter:

```
isql -U sa -P -S Kerberos_PDS
use master
go
sp_configure 'use security services', 1
go
```

```
isql -U sa -P -S Kerberos_RDS
use master
go
sp_configure 'use security services', 1
go
```

6. Restart Adaptive Servers with Kerberos security mechanism.

To restart the *Kerberos_PDS*:

```
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
export KRB5_KTNAME=/usr/Kerberos_PDS_principal.keytab
export KRB5CCNAME=/usr/Kerberos_PDS_principal.cc
export KRB5_CONFIG=/mitkrb/krb5.conf
export SYBASE_PRINCIPAL=Kerberos_PDS_principal@ASE //
use SYBASE_PRINCIPAL or "-k" option
dataserver -d data_Kerberos_PDS.dat -k Kerberos_PDS_principal@ASE
-s Kerberos_PDS -c Kerberos_PDS.cfg
//add "-k Kerberos_PDS_principal@ASE"
```

To restart the *Kerberos_RDS*:

```
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
export KRB5_KTNAME=/usr/Kerberos_RDS_principal.keytab
export KRB5CCNAME=/usr/Kerberos_RDS_principal.cc
export KRB5_CONFIG=/mitkrb/krb5.conf
export SYBASE_PRINCIPAL=Kerberos_RDS_principal@ASE //
use SYBASE_PRINCIPAL or "-k" option
dataserver -d data_Kerberos_RDS.dat -k Kerberos_RDS_principal@ASE
-s Kerberos_RDS -c Kerberos_PDS.cfg
//add "-k Kerberos_PDS_principal@ASE"
```

7. Using sa user credentials, access Adaptive Servers when Kerberos is on:

```
cd $SYBASE
. SYBASE.sh
export KRB5_CONFIG=/mitkrb/krb5.conf
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
export PATH=/bin:$PATH
/bin/kinit sa
Password for sa@ASE: password
isql -V -I interfaces -J -U sa -S Kerberos_PDS_principal -w 700
isql -V -I interfaces -J -U sa -S Kerberos_RDS_principal -w 700
```

Manage Replication Server Security

where:

- **isql** – is the Interactive SQL parser to Replication Server.
- **-V** – specifies the network-based security options for user authentication.
- **-S** – specifies the server's network name.
- **-U** – specifies the user name.
- **-P** – specifies the password for the user name.

Configuring Replication Server to Use Kerberos

Configure the Replication Server to use a user-defined Kerberos principal name to replicate data from an Adaptive Server database.

In this example, you configure the primary Replication Server (*PRS*) principal name as *PRS_principal* and the replicate Replication Server (*RRS*) principal name as *RRS_principal*.

1. Follow step 1 to step 4 in the '*Configuring Adaptive Server to Use Kerberos*' to add *PRS_principal* and *RRS_principal* in the KDC, interfaces, and libtcl64.cfg configuration files.

Enter Replication Server principal names in the `interfaces` file as:

```
../$ vi interfaces
PRS
    query tcp ether replinuxb15 30002
    master tcp ether replinuxb15 30002
    secmech 1.3.6.1.4.1.897.4.6.6
PRS_principal
    query tcp ether replinuxb15 30002
    master tcp ether replinuxb15 30002
    secmech 1.3.6.1.4.1.897.4.6.6
RRS
    query tcp ether replinuxb15 30003
    master tcp ether replinuxb15 30003
    secmech 1.3.6.1.4.1.897.4.6.6
RRS_principal
    query tcp ether replinuxb15 30003
    master tcp ether replinuxb15 30003
    secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_PDS
    query tcp sun-ether replinuxb15 20001
    master tcp sun-ether replinuxb15 20001
    secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_PDS_principal
    query tcp sun-ether replinuxb15 20001
    master tcp sun-ether replinuxb15 20001
    secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_RDS
    query tcp sun-ether replinuxb15 20002
    master tcp sun-ether replinuxb15 20002
    secmech 1.3.6.1.4.1.897.4.6.6
Kerberos_RDS_principal
    query tcp sun-ether replinuxb15 20002
```



```
master tcp sun-ether replinxb15 20002
secmech 1.3.6.1.4.1.897.4.6.6
```

Note: Make sure both *Kerberos_PDS*, and *Kerberos_RDS* and their principal names are also included in the `interfaces` file. Adaptive Server `interfaces` file should contain the same information as Replication Server `interfaces` file.

2. In Adaptive Server, add Replication Server principal names:

```
isql -U sa -P -S Kerberos_PDS
use master
go
sp_addlogin PRS_principal,a1234abcd
go
sp_adduser PRS_principal
go
grant set session authorization to PRS_principal
// or: grant set proxy to PRS_principal
go
```

```
isql -U sa -P -S Kerberos_RDS
use master
go
sp_addlogin RRS_principal,a1234abcd
go
sp_adduser RRS_principal
go
grant set session authorization to RRS_principal
// or: grant set proxy to RRS_principal
go
```

3. Add Replication Server users and their principal names:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
create user RRS_principal set password a1234abcd
go
grant sa to RRS_principal
go
grant connect source to RRS_principal
go
```

```
isql -U sa -P a1234abcd -S RRS -I interfaces -J -w 700
create user PRS_principal set password a1234abcd
go
grant sa to PRS_principal
go
grant connect source to PRS_principal
go
```

To access Replication Server with the Replication Agent thread, enter:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
create user Kerberos_PDS_principal set password a1234abcd
go
grant sa to Kerberos_PDS_principal
go
```

Manage Replication Server Security

```
grant connect source to Kerberos_PDS_principal
go
```

4. If you use a local mechanism name other than the default name, configure the `objectid.dat`.

Add the local name in the `objectid.dat` file and ensure it includes this line:

```
"1.3.6.1.4.1.897.4.6.6 = csfkrb5"
```

5. To configure **use security services**, enter.

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
configure replication server set 'use_security_services' to 'on'
go
shutdown
```

```
isql -U sa -P a1234abcd -S RRS -I interfaces -J -w 700
configure replication server set 'use_security_services' to 'on'
go
shutdown
```

See *Replication Server Reference Manual* > *Replication Server Commands* > *configure replication server*.

6. Restart the Replication Servers.

To restart *PRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH

export KRB5_KTNAME=/usr/PRS_principal.keytab

export KRB5CCNAME=/usr/PRS_principal.cc

export SYBASE_RS_PRINCIPAL=PRS_principal

export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_PRS
```

To restart *RRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH

export KRB5_KTNAME=/usr/RRS_principal.keytab

export KRB5CCNAME=/usr/RRS_principal.cc

export SYBASE_RS_PRINCIPAL=RRS_principal

export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_RRS
```

7. Configure the **security_mechanism** and **ID_security_mechanism** to use Kerberos.

For *PRS*:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
configure replication server set 'security_mechanism' to 'csfkrb5'
```

```
go
configure replication server set 'id_security_mechanism' to
'csfkrb5'
go
shutdown
```

For *RRS*:

```
isql -U sa -P a1234abcd -S RRS -I interfaces -J -w 700
configure replication server set 'security_mechanism' to 'csfkrb5'
go
configure replication server set 'id_security_mechanism' to
'csfkrb5'
go
shutdown
```

8. Restart *PRS* and *RRS*:

To restart *PRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH

export KRB5_KTNAME=/usr/PRS_principal.keytab

export KRB5CCNAME=/usr/PRS_principal.cc

export SYBASE_RS_PRINCIPAL=PRS_principal

export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_PRS
```

To restart *RRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH

export KRB5_KTNAME=/usr/RRS_principal.keytab

export KRB5CCNAME=/usr/RRS_principal.cc

export SYBASE_RS_PRINCIPAL=RRS_principal

export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_RRS
```

9. Configure the *unified_login*, and *id_unified_login* settings.

For *PRS*:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
configure replication server set unified_login to 'required'
go
configure replication server set id_unified_login to 'required'
go
shutdown
```

For *RRS*:

Manage Replication Server Security

```
isql -U sa -P a1234abcd -S RRS -I interfaces -J -w 700
configure replication server set unified_login to 'required'
go
configure replication server set id_unified_login to 'required'
go
shutdown
```

See *Replication Server Reference Manual* > *Replication Server Commands* > *configure replication server*.

10. Add the RSSD details in the Replication Server configuration file.

For *PRS*:

```
RSSD_unified_login=required
#RSSD_mutual_auth=not_required
#RSSD_msg_confidentiality=not_required
#RSSD_msg_integrity=not_required
#RSSD_msg_origin_check=not_supported
#RSSD_msg_replay_detection=not_required
#RSSD_msg_sequence_check=not_required
RSSD_sec_mechanism=csfkrb5
RSSD_server_principal=Kerberos_PDS_principal // this is for PRS
ID_server_principal=PRS_principal // if this RS is not ID server.
```

For *RRS*:

```
RSSD_unified_login=required
#RSSD_mutual_auth=not_required
#RSSD_msg_confidentiality=not_required
#RSSD_msg_integrity=not_required
#RSSD_msg_origin_check=not_supported
#RSSD_msg_replay_detection=not_required
#RSSD_msg_sequence_check=not_required
RSSD_sec_mechanism=csfkrb5
RSSD_server_principal=Kerberos_RDS_principal // this is for RRS
ID_server_principal=PRS_principal // if this RS is not ID server.
```

Add principal user names in the `rs_principal_users.cfg` configuration file, if needed.

11. Configure the **unified_login** settings in Adaptive Server.

For *Kerberos_PDS*:

```
isql -U sa -P -S Kerberos_PDS
sp_configure "unified login required", 1
```

For *Kerberos_RDS*:

```
isql -U sa -P -S Kerberos_RDS
sp_configure "unified login required", 1
```

12. Restart *PRS* and *RRS* with Kerberos security mechanism:

To restart *PRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
```

```
export KRB5_KTNAME=/usr/PRS_principal.keytab
export KRB5CCNAME=/usr/PRS_principal.cc
export SYBASE_RS_PRINCIPAL=PRS_principal
export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_PRS
```

To restart *RRS*:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
export KRB5_KTNAME=/usr/RRS_principal.keytab
export KRB5CCNAME=/usr/RRS_principal.cc
export SYBASE_RS_PRINCIPAL=RRS_principal
export KRB5_CONFIG=/mitkrb/krb5.conf/
REP-15_5/install/RUN_RRS
```

- 13.** If the Replication Server and Adaptive Server principal names are different from their server names, add principal user names in the `rs_principal_users.cfg` configuration file.

Add one line for each Replication Server and Adaptive Server:

- PRS:PRS_principal
- RRS:RRS_principal
- Kerberos_PDS:Kerberos_PDS_principa
- Kerberos_RDS:Kerberos_RDS_principal

- 14.** If you do not want to restart Replication Server, reload the principal user names from the `rs_principal_users.cfg`:

```
sysadmin principal_users, reload
```

You can also use this command to display all principal names.

- 15.** To access *PRS* with the Replication Agent thread, enter:

```
isql -V -I interfaces -J -U sa -S Kerberos_PDS_principal
use PRS_RSSD
go
sp_config_rep_agent PRS_RSSD, 'security mechanism', 'csfkrb5'
go
sp_config_rep_agent PRS_RSSD, 'unified login', 'true'
go
sp_config_rep_agent PRS_RSSD, 'rs_password', 'NULL'
go
sp_config_rep_agent PRS_RSSD, 'rs servername', 'PRS_principal'
go
sp_stop_rep_agent PRS_RSSD
go
sp_start_rep_agent PRS_RSSD
go
```

Manage Replication Server Security

```
use prspdb
go
sp_config_rep_agent prspdb, 'security mechanism', 'csfkrb5'
go
sp_config_rep_agent prspdb, 'unified login', 'true'
go
sp_config_rep_agent prspdb, 'rs_password', 'NULL'
go
sp_config_rep_agent prspdb, 'rs servername', 'PRS_principal'
go
sp_stop_rep_agent prspdb
go
sp_start_rep_agent prspdb
go
```

16. Access Replication Server when Kerberos security is on.

To access *PRS*, using sa user credentials:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J -w 700
isql -U sa -P a1234abcd -S PRS_principal -I interfaces -J -w 700
```

To access *RRS*, using sa user credentials:

```
isql -U sa -P a1234abcd -S RRS -I interfaces -J -w 700
isql -U sa -P a1234abcd -S RRS_principal -I interfaces -J -w 700
```

Or, access Replication Server with the principal user credential:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/mitkrb/linux/lib:$LD_LIBRARY_PATH
export KRB5_CONFIG=/mitkrb/krb5.conf
/mitkrb/linux/bin/kinit sa
Password for sa@ASE: password

isql -V -I interfaces -J -w 700 -U sa -S PRS_principal
isql -V -I interfaces -J -w 700 -U sa -S RRS_principal
```

17. Use non-sa user credentials to access Replication Server when Kerberos security is on.

a) To access Replication Server with the user name 'scott', add user scott in KDC:

```
export LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH
export KRB5_CONFIG=/mitkrb/krb5.conf
export PATH=/mitkrb/linux/bin:$PATH
/sbin/kadmin -p kadmin/admin -k -t mitkrb/linux/admin.key
addprinc scott@ASE // 1234abcd
ktadd -k /usr/u/scott/scott.keytab scott@ASE
quit

export KRB5CCNAME=/usr/u/scott/scott.cc
/mitkrb/linux/bin/kinit -f -k -t /usr/u/scott/scott.keytab
scott@ASE
```

b) Add user 'scott' in Replication Server:

```
isql -U sa -P a1234abcd -S PRS -I interfaces -J
create user scott@ASE set password a1234abcd
```

c) Access Replication Server, either with the user name and password:

```
isql -U scott -P a1234abcd -S PRS -I interfaces -J
```

Or, with the principal user credentials:

```
cd $SYBASE
. SYBASE.sh
export LD_LIBRARY_PATH=/mitkrb/linux/lib:$LD_LIBRARY_PATH
export KRB5_CONFIG=/mitkrb/krb5.conf
export KRB5CCNAME=/usr/u/scott/scott.cc
isql -V -S PRS_principal -I interfaces
//if your login is "scott"
isql -V -U scott -S PRS_principal -I interfaces
```

Follow these same steps to configure a single Adaptive Server which acts both the primary and the replicate database in an Adaptive Server-to-Adaptive Server replication system.

Establish the Principal User

When network-based security is enabled with unified login, Replication Server must log in to remote servers as the principal user.

When network security is not enabled, Replication Server logs in to remote servers as one of several possible users, depending on the task to be performed. The principal user credential is the only credential Replication Server has to log in to other processes when network security is active.

When Replication Server logs in to another Replication Server or a data server, the principal user name contained in the credential is mapped to the server name space and a secure connection is established.

Note: Make sure that principal user names are unique. Replication Server cannot log in to another server of the same name.

Replication Server executes the **set proxy** command in the remote server (as the principal user) and switches to the appropriate user for the current task.

Identify the Principal User

You can specify the principal user name with the **-S** flag when you log in to or start Replication Server.

It is the responsibility of the replication system administrator to establish a principal user for each Replication Server. Sybase recommends that you use the name of the Replication Server as the principal user name.

If you do not specify a principal user name using the **-S** flag, Replication Server uses the Replication Server name.

Identify the Principal User to the Security Mechanism

The security administrator for the security mechanism must define the Replication Server principal name to the security mechanism.

For DCE:

- Use the DCE **dcecp** tool's **user create** command to create the principal user.
When you are defining a server to DCE, use options that specify that the new principal user can act as a server.
- Use the **keytab create** command of the **dcecp** utility to create a DCE key table file, which contains a principal user's password in encrypted form.

Note: DCE is not supported on UNIX

For CyberSafe Kerberos:

- Use the Kerberos **csfadml** tool to create the principal user.
- Use **csfadml** to extract the key table file.

For MIT Kerberos:

- Use administrative command **addprinc** to create principal user.
- Use administrative command **ktadd** to extract key table file.

Refer to documentation from the security mechanism provider for detailed information about identifying servers and users to the security mechanism.

Identify Principal Users to Replication Server

The principal user for other processes—including RepAgents, data servers, and other Replication Servers—using system security and unified login to connect to Replication Server must be identified in the `rs_users` table for the current Replication Server. You can use the **create user** command to add principal user names to `rs_users`.

Identify the Replication Server Principal User to the Replication System

You must add the Replication Server principal user name to destination processes—Replication Servers and data servers—including the ID Server and the RSSD to which Replication Server is connecting using unified login.

Refer to the *Adaptive Server Enterprise System Administration Guide* for information about adding login names to Adaptive Server.

Activating Network-based Security

Before configuring security services, you must activate network-based security for the Replication Server using the **configure replication server** command.

1. Log in to Replication Server and enter:


```
configure replication server
  set use_security_services to 'on'
```

2. Shut down Replication Server.
3. Restart Replication Server by executing the **repserver** command or the Replication Server run file.
 - If you are using the DCE security mechanism, make sure you include the **-K** flag to specify the key table file location.
 - If you are using the Kerberos security mechanism, the key table location must be specified by the `KRB5_KTNAME` environment variable (UNIX) or the key table registry key entry (Windows 2000 or 2003).

Refer to the *Replication Server Reference Manual* for syntax and other information about the **repserver** command.

See also

- *Disabling Network-based Security* on page 293

Start Server and Clients

For the network security environment to work properly, both servers and clients should be started only after they have a valid credential.

For Kerberos systems:

- On UNIX systems, servers and clients should be started after a **kinit**
- On Windows NT systems, server and clients can be started automatically using the single sign-on feature or manually using the Kerberos credentials manager.

Refer to your Kerberos documentation for more information.

Transarc DCE systems behave in similar manner, refer to your Transarc documentation for information about setting up the proper environment.

Configure Security Services for Replication Server

Learn what network-based security features Replication Server parameters provide.

Configuration parameters enable:

- Unified login
- Mutual authentication
- Choice of supported security mechanism
- Message confidentiality through encryption
- Other secure message transmission features: message integrity, origin check, replay detection, and out-of-sequence detection

Note: Depending on the security mechanism you choose, one or more of these security features may not be available at your site.

Manage Replication Server Security

You set default parameters in the **rs_init** program during system configuration. Refer to the *Replication Server Configuration Guide* for your platform for information about **rs_init**. This section describes how you set these parameters at the command line.

Identify Replication Server Pathways

Replication Server coordinates data replication activities for local data servers and exchanges data with Replication Servers and data servers at other sites. Learn the Replication Server pathways for which you can configure network-based security.

- When Replication Server is acting as a client, you can configure security for:
 - All pathways established when Replication Server logs in to another server. These are default global settings.
 - The connection to the RSSD.
 - Individual connections.
 - Individual routes.
 - Replication Server to ID Server pathway.
 - Pathways used to create a route, create a subscription, or drop a subscription.
- When Replication Server is acting as a server, you can configure security for:
 - All incoming logins. These are default global settings.
 - User connection to Replication Server (set when logging on).

Secure Different Types of Network Pathway

Configure security for the different types of pathway for outgoing logins.

Table 19. Network Pathways

Pathway	How to Secure It	Special Parameters and Exceptions
All pathways initiated by the current Replication Server (acting as a client)	Set global security parameters using configure replication server . This is the default setting for all outgoing logins unless overridden for individual pathways.	Use use_security_services to turn off all network security with a single command.
Connection to the RSSD	Use a text editor to configure the <code>rs_config</code> file.	Security parameters have an “RSSD_” prefix. For example: RSSD_unified_login .

Pathway	How to Secure It	Special Parameters and Exceptions
Individual connections	Set security parameters for a connection to a remote database with: <ul style="list-style-type: none"> • create connection, or • alter connection See the <i>Replication Server Reference Manual</i> for more information about these commands.	Use dsi_exec_request_sproc to suspend request stored procedures.
Individual routes defined using the create route command	Set security parameters using: <ul style="list-style-type: none"> • create route, or • alter route See the <i>Replication Server Reference Manual</i> for more information about these commands.	
Replication Server to ID Server	Set security parameters with configure replication server . See the <i>Replication Server Reference Manual</i> for more information about this command.	Security parameters have an “id-” prefix. For example: id_msg_confidentiality .
Replication Server to primary Replication Server and primary database to: <ul style="list-style-type: none"> • create a route • create or drop a subscription 	Replication Server duplicates the security settings used when the user creating the route or creating or dropping the subscription logs in to Replication Server.	
All incoming logins (Replication Server acting as server)	Set parameters for incoming logins with configure replication server . Default parameters for outgoing and incoming parameters are set at the same time and are identical.	
Pathway established when user logs in to Replication Server.	Set security parameters with the isql utilities.	Security parameters set for this pathway must be compatible with those set at the Replication Server for all incoming logins. Security for this pathway cannot be configured using the rs_init utility.

See also

- *Configure Security for Database Connections* on page 286
- *Replication Server Borrows Security Settings to Secure Other Pathways* on page 292
- *Disabling Network-based Security* on page 293

Security Configuration Parameters

Replicator Server provides security configuration parameters generally available for all pathways.

Table 20. Security Parameters Affecting Replication Server

configuration_parameter	Description
msg_confidentiality	Indicates whether Replication Server sends and receives encrypted data. If set to “required,” outgoing and incoming data must be encrypted. If set to “not_required,” Replication Server accepts incoming data that is encrypted or not encrypted. Values are “required” or “not_required.” Default: not_required
msg_integrity	Indicates whether data is checked for tampering. Values are “required” or “not_required.” Default: not_required
msg_origin_check	Indicates whether the source of data must be verified. Values are “required” or “not_required.” Default: not_required
msg_replay_detection	Indicates whether data should be checked to make sure it has not been intercepted and re-sent. Values are “required” or “not_required.” Default: not_required
msg_sequence_check	Indicates whether data packages should be checked to ensure that they have been received in the order sent. Values are “required” or “not_required.” Default: not_required
mutual_auth	Requires remote server to provide proof of identify before a connection can be established. Values are “required” or “not_required.” Default: not_required
security_mechanism	Specifies the name of the network-based security mechanism. Default: First security mechanism listed in <code>libtcl.cfg</code> .

<i>configuration_parameter</i>	Description
unified_login	<p>Indicates how Replication Server seeks outgoing connections and accepts incoming connections. The values are:</p> <ul style="list-style-type: none"> “required” – always seeks to log in to remote server with a credential; only accepts incoming logins with a credential. “not_required” – always seeks to log in to remote server with a password; accepts incoming logins with a credential or a password. <hr/> <p>Note: unified_login must be “required” before other security parameters can take effect.</p> <hr/> <p>Default: not_required</p>

Plan for Compatible Settings

When you set up network-based security, you must plan for the interaction between security settings at each end of the secured pathway. Security settings at both ends of each pathway must be compatible.

Replication Server accepts incoming logins and initiates logins to other servers. Security parameters for all incoming logins (when Replication Server is acting as a server) are set with the **configure replication server** command. There are also security parameters you set for outgoing logins (when Replication Server is acting as a client) that you set for network pathways.

Note: It is the responsibility of the replication system administrator 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 pathway. An attempted login fails if security features at both ends of the pathway are not compatible.

Use **configure replication server** with the **use_security_services** parameter to activate or deactivate all compatible security settings for client/server interaction. If the security services parameters are not compatible, for example, if a parameter is set to “not_required” at the client and “required” at the server, the server does not allow the client to log in.

See also

- *Secure Different Types of Network Pathway* on page 280

Compatible Client/Server Settings

Use the server settings that are compatible with the client settings.

Table 21. Compatible Client/Server Settings

Client	Server
use_security_services “off”: no security services	Compatible settings: <ul style="list-style-type: none"> • use_security_services “off”, or • use_security_services “on” and security feature “not required”
use_security_services “on” and security feature “not required”	Compatible settings: <ul style="list-style-type: none"> • use_security_services “on” and security feature “not required,” or • use_security_services “off”
use_security_services “on” and security feature “required”	Compatible settings: <ul style="list-style-type: none"> • use_security_services “on” and security feature “required”

Configure Default Values

Use **configure replication server** to establish default security settings for all outgoing logins (when Replication Server acts as a client) and incoming logins (when Replication Server acts as a server).

You can override default security settings for these outgoing pathways:

- Individual connections
- Individual routes
- The pathway from Replication Server to ID Server

Note: You cannot override any default security settings that control security for incoming logins.

When Replication Server seeks to open a pathway to another server, it checks to see if security parameters have been set specifically for that pathway. If not, Replication Server uses the default security settings determined using **configure replication server**.

To set global security parameters, log in to Replication Server and execute **configure replication server** at the **isql** prompt. Here is the syntax:

```
configure replication server {
  set security_mechanism to 'mechanism_name' |
  set security_parameter to { 'required' |
    'not_required' }}
```

You can set all of the security configuration parameters affecting Replication Server. They are stored in the `rs_config` table in the RSSD. You must have **sa** permission to execute them.

Require Unified Login

To require all servers and users that connect to Replication Server to be authenticated by the security mechanism, set **unified_login** to 'required'.

Log in to Replication Server and execute this command at the **isql** prompt:

```
configure replication server
  set unified_login to 'required'
```

If **unified_login** is 'not_required', Replication Server allows servers and users to connect with either a credential or a password.

Note: **unified_login** must be 'required' for other security services to take effect.

Require Data Encryption

To require all data sent or received by Replication Server to be encrypted, set **msg_encryption** to 'required'.

Log in to the Replication Server and execute this command at the **isql** prompt:

```
configure replication server
  set msg_encryption to 'required'
```

See also

- *Configure Security for Database Connections* on page 286
- *Configure Security for Routes* on page 288
- *Configure Security to the ID Server* on page 289

Configure Security for the Connection to the RSSD

At startup, Replication Server contacts the RSSD for configuration information. Secure the pathway from Replication Server to RSSD using network-based security.

When you set up Replication Server, **rs_init** creates the RSSD connection and places default security information in the Replication Server configuration file, `Rep_Server_name.cfg`. By default, **rs_init** sets all network security parameters to “not required.” If you want to secure the pathway, you must use a text editor to change desired default values to “required.”

Configuration values for the RSSD are preceded by an “RSSD_” prefix. For example:

- **RSSD_mutual_auth**
- **RSSD_msg_origin_check**
- **RSSD_server_principal_name**

For example, to set RSSD server principal name to *Kerberos_PDS_princ*, add this line in the `Rep_Server_name.cfg` configuration file:

Manage Replication Server Security

```
RSSD_server_principal=Kerberos_PDS_princ
```

Note: If you do not enter **RSSD_server_principal_name** in the *Rep_Server_name.cfg* configuration file, Replication Server considers that RSSD server principal name is same as RSSD server name.

See also

- *Security Configuration Parameters* on page 282

Configure Security for Database Connections

Use **create connection** or **alter connection** to configure security for individual connections.

Security parameters configured with these commands affect security for the outgoing connection to the data server. They override parameters set with **configure replication server**.

Create a Secure Connection

You can set security parameters when you create a connection with **create connection**.

Normally, you use **create connection** to add connections to non-Sybase databases.

Here is the syntax for including security features with the **create connection** command. See *Replication Server Reference Manual > Replication Server Commands > create connection* for detailed information about using **create connection**.

```
create connection to data_server.database...
  set username [to] user
  [set password [to] passwd]
  [set security_mechanism [to] 'mechanism_name' |
  set dsi_exec_request_sproc [to] { 'on' | 'off' } |
  set security_mechanism [to] 'mechanism_name' |
  set security_parameter [to] { 'required' |
  'not_required' } ]
```

In addition to security configuration parameters you can set with **create connection**, you can set the **dsi_exec_request_sproc** special security parameters.

Connections parameters are stored in the *rs_config* table in the RSSD, and you must have **sa** permission to execute them.

Security parameters set at both ends of a connection must be compatible.

See also

- *Security Configuration Parameters* on page 282

Special Security Parameters for Connections

You can use special parameters to secure connections.

Table 22. Special Security Parameters for Connections

<i>security_parameter</i>	Description
dsi_exec_request_sproc	Indicates whether request stored procedures at the primary Replication Server are “off” or “on.” Use in multiple security-system environments. Default: off

See also

- *Plan for Compatible Settings* on page 283
- *Use More than One Security Mechanism* on page 295

Modifying Security for a Connection

Use **alter connection** to change the security settings for a database connection.

Syntax:

```
alter connection to data_server.database {
...
set password to passwd |
set security_mechanism to 'mechanism_name' |
set dsi_exec_request_sproc to { 'on' | 'off' } |
set security_parameter to { 'required' |
'not_required' }}
```

To change the security parameters of a database connection, perform these steps at the Replication Server:

1. Execute **suspend connection** to suspend activity on the connection.
2. Execute **alter connection** to change a security parameter.

Note: Set one parameter at a time.

3. Execute **resume connection** to resume activity on the connection.

Examples of Using **alter connection** to Configure Security

- To require Replication Server to connect to the target database (TOKYO_DS.pubs2) with a credential, execute:

```
alter connection to TOKYO_DS.pubs2
set unified_login to 'required'
```

Note: **unified_login** must be “required” for other security services to take effect.

- To turn “off” request stored procedures at the TOKYO data server in a multiple security-system environment, execute:

Manage Replication Server Security

```
alter connection to TOKYO_DS.pubs2
  set dsi_exec_request_sproc to 'off'
```

See also

- *Security Configuration Parameters* on page 282
- *Special Security Parameters for Connections* on page 287

Configure Security for Routes

Use **create route** or **alter route** to configure security for individual routes.

Security parameters configured with these commands affect security for the outgoing login to the destination Replication Server. They override default parameters set with **configure replication server**.

Create a Secure Route

You can set security parameters when you create a route with **create route**.

Here is the syntax for including security features using the **create route** command:

```
create route to dest_replication_server {
...
[set username to 'user' ]
[set password to 'passwd' ]
[set security_mechanism to 'mechanism_name' |
set security_parameter to { 'required' |
  'not_required' } ]
```

The parameters are stored in the `rs_config` table in the RSSD. You must have **sa** permission to execute them.

Security parameters set at both ends of a route must be compatible.

See also

- *Plan for Compatible Settings* on page 283
- *Security Configuration Parameters* on page 282

Modifying Security for a Route

Use **alter route** to change the security settings for a route.

Here is the syntax for altering security:

```
alter route to dest_replication_server {
...
set password to 'passwd' |
set security_mechanism to 'mechanism_name' |
set security_parameter to { 'required' |
  'not_required' }}
```

To change the security parameters of a route, log in to Replication Server and perform these steps at the Replication Server:

1. Execute **suspend route** to suspend activity on the route.
2. Execute **alter route** to change a security parameter.

Note: Set one parameter at a time.

3. Execute **resume route** to resume activity on the route.

Examples of Using `alter route` to Configure Security

- To require Replication Server to connect to the target Replication Server (TOKYO_RS) with a password, execute these commands:

```
alter route to TOKYO_RS
  set username 'TOKYO_rsi_user'
```

```
alter route to TOKYO_RS
  set password 'TOKYO_rsi_pw'
```

```
alter route to TOKYO_RS
  set unified_login to 'not_required'
```

Note: If `unified_login` is “not_required,” you must specify an RSI user and password.

- To specify that all messages exchanged with the target Replication Server (TOKYO_RS) are checked for tampering, execute:

```
alter route to TOKYO_RS
  set msg_integrity to 'required'
```

See also

- *Security Configuration Parameters* on page 282

Configure Security to the ID Server

Use **configure replication server** to configure network-based security for the network connection from Replication Server to ID Server.

The syntax is:

```
configure replication server
  set id_security_param to { 'required' |
    'not_required' }
```

See the *Replication Server Reference Manual > Replication Server Commands > **configure replication server***. The security parameters are stored in the `rs_config` table in the RSSD. You must have **sa** permission to configure them. To distinguish settings for this pathway, all ID Server parameters begin with the “id_” prefix. For example:

- **id_msg_confidentiality**
- **id_security_mechanism**

ID Server security parameters configured with **configure replication server** are dynamic. They take effect immediately and do not require that you restart Replication Server.

Examples of Using configure replication server to Configure Security to the ID Server

- To require that the source of all messages be verified, log in to the source Replication Server and enter:

```
configure replication server
  set id_msg_origin_check 'required'
```

- To require that Replication Server logs in to ID Server with a credential, enter:

```
configure replication server
  set id_unified_login to 'required'
```

See also

- Security Configuration Parameters* on page 282

Log in to Replication Server

Connect to Replication Server using a client application such as **isql** or a custom application program you create with Open Client Client-Library.

isql Command Line Options for Security

There are several command line options that you can use with **isql** to enable network-based security services for the connection to Replication Server.

Table 23. isql Command Line Options for Security

Option name	Meaning
-K <i>keytab_file</i>	Use only with DCE security. It specifies a DCE keytab file that contains the security key for the user logging into the server. Keytab files can be created with the DCE dcecp utility—see your DCE documentation for more information. Replication Server must have read permission on this file. Note: For Kerberos users: Specify the location of the key table file using the key table registry key entry (Windows 2000 or 2003).
-S <i>server_name</i>	Specifies the server's network name. If unified login is enabled, this option also specifies the principal user.

Option name	Meaning
-V <i>security_options</i>	<p>Specifies unified login. With this option, the user must log in to the network's security system before running the isql utility. If a user specifies the -U option, the user must supply the network user name known to the security mechanism; any password supplied with the -P option is ignored.</p> <p>-V can be followed by a string of options that enable additional security services. Here is a list of options and the services they enable.</p> <ul style="list-style-type: none"> • c – data confidentiality • i – data integrity • m – mutual authentication • o – data origin stamping service • r – data replay detection • q – out-of-sequence detection
-X	Specifies that connections are made with encrypted passwords.
-Z <i>security_mechanism</i>	<p>Specifies the name of a security mechanism to use on the connection to Replication Server.</p> <p>Supported security mechanism names are listed in the <code>libtcl.cfg</code> file. If no security mechanism is supplied, the default is used, which is the first security mechanism listed under SECURITY in <code>libtcl.cfg</code>.</p>

Examples of Connecting to Replication Server

You can connect to Replication Server by logging in to the security mechanism and then logging in to Replication Server, or you can log directly in to Replication Server.

You must include the **-S** flag to identify the principal user.

Connecting to Replication Server from the Security Mechanism

Example that shows how to log in first to the DCE security mechanism and then to Replication Server.

Note: When using DCE, if you want to log in as another user, you must include the **-U** and **-K** options.

1. Log in to the DCE/Kerberos security mechanism and receive a credential:

- For DCE, enter

```
dce_login user_name password
```
- For Kerberos, enter

```
kinit user_name password
```

2. Log in to Replication Server with **isql**:

- For DCE, enter

Manage Replication Server Security

```
isql -Srs_server_name -Vsecurity_option
```

- For Kerberos, enter

```
isql -Srs_server_principal_name -Vsecurity_option
```

Connecting to Replication Server from Outside Security

Example that shows how to connect to Replication Server from outside the security mechanism.

Enter:

- For DCE:

```
isql -Srs_server_name -User_name  
-Kkeytab_file
```

- For Kerberos:

```
isql -Srs-server_name -Yuser_name
```

Replication Server Borrows Security Settings to Secure Other Pathways

The security services you use when logging in to Replication Server from the command line not only secure the pathway between the client and the server, they may also be duplicated later on in the session when Replication Server opens other pathways.

Replication Server logs in to the primary Replication Server and the primary database over informal pathways when executing these commands:

- **create subscription**
- **drop subscription**
- **create route**

To secure these pathways, Replication Server borrows the security settings entered by the user executing **create subscription**, **drop subscription**, or **create route** when that user logged in to Replication Server.

Maintain Network Security

Learn to manage and maintain network security.

Use set proxy To Switch Logins

Use **set proxy** to allow Replication Server commands to be executed on the target data server by the correct user.

When **unified_login** is enabled, Replication Server always logs in to remote processes as the principal user. Nonetheless, Replication Server commands must be executed on the target data server by the correct user for a particular operation. For example, Replication Server must use the maintenance user login name when applying changes to replicate databases. Replication Server uses the Adaptive Server **set proxy** command to switch automatically from the login user to the required user.

You can customize the **set proxy** command with the **rs_setproxy** function string. **rs_setproxy** changes the login name in a data server. **rs_setproxy** has function-string-class scope.

See *Replication Server Reference Manual > Replication Server System Functions > rs_setproxy* and *Replication Server Reference Manual > Replication Server Commands > set proxy*.

Disabling Network-based Security

Use the **use_security_services to 'off'** parameter to disable all security services.

When you disable network security, Replication Server does not accept incoming logins with security credentials and does not attempt to log in to other processes with a security credential. No security services are active.

1. Log in to the Replication Server, and at the **isql** prompt enter this command:

```
configure replication server
    set use_security_services to 'off'
```

2. Restart Replication Server by executing the **repserver** command or the Replication Server run file.

See also

- *Replication Server Executable Program* on page 88
- *Set up Network-based Security* on page 263

Change the Security Mechanism

Use the **set security_mechanism** parameter to change to another security mechanism.

Log in to Replication Server and execute this command at the **isql** prompt:

```
configure replication server
    set security_mechanism to 'mechanism_name'
```

The security mechanism you change to must be installed and listed in the SECURITY section of the `libtcl.cfg` file.

Reset Per-target Values to Default Values

You can change all of your per-target security values for connections or routes using the **set security_services to 'default'** option.

Resetting Per-target Values for Connections

Use **alter connection** to change per-target security settings to global values set with **configure replication server**.

Changes take effect after you resume the connection. This procedure does not affect the configuration of **use_security_services**.

1. Log in to the Replication Server and execute **suspend connection** at the **isql** prompt. Enter:

```
suspend connection to dataserver.database
```

Manage Replication Server Security

2. Change security settings with **alter connection**. Enter:

```
alter connection to dataserver.database
  set security_services to 'default'
```

3. Resume the route or connection with **resume connection** for the changes to take effect. Enter:

```
resume connection to dataserver.database
```

Resetting Per-target Values for Routes

Use the **set security_services to 'default'** parameter with **alter route** to change per-target security settings to global values set with **configure replication server**.

Changes take effect after you resume the route. This procedure does not affect the configuration of **use_security_services**.

1. Suspend the route. Enter:

```
suspend route to dest_replication_server
```

2. Alter the route. Enter:

```
alter route to dest_replication_server
  set security_services to 'default'
```

3. Resume the route. Enter:

```
resume route to dest_replication_server
```

View Information About Security Services

You can display information about the Replication Server network-based security using the **admin security_property** and **admin security_setting** commands.

What Security Mechanisms and Services Are Available?

Use **admin security_property** to find out which security mechanisms and services are supported at the Replication Server:

```
admin security_property[, security_mechanism]
```

Replication Server displays the name of supported security mechanisms, the security services available for that mechanism, and whether or not those services are supported at your site.

What Are the Current Security Settings?

Use **admin security_setting** to determine the status of supported security services.

You can view status information for security parameters that have been set for routes and/or the ID Server. You can use the following syntax, where *rs_idserver* is the name of the ID Server and *rep_server* is the name of the destination Replication Server:

```
admin security_setting[, rs_idserver |, rep_server ]
```


How Replication Server Maps a Security System Login to a Replication Server Login

Learn how Replication Server maps a security system login to a Replication Server login and how Replication Server translates invalid characters.

Your network-based security mechanism may use login names that are not valid on Replication Server. For example, login names on Replication Server must not exceed 30 characters or include certain special characters such as *, (, and %. Login names on Replication Server must be valid identifiers, which are described in *Replication Server Reference Manual > Topics > Identifiers*.

If the security login name is not a valid identifier, Replication Server automatically maps invalid characters to valid characters and truncates login names at 30 characters.

How Replication Server Converts Invalid Characters

Learn the mapping of invalid characters to valid characters when Replication Server translates invalid characters.

Table 24. Conversion of Invalid Characters to Valid Characters

Invalid characters	Convert to
\% & , : = > ' ' ~	an underscore: _
! ^ () . < ? { }	a dollar sign: \$
“ - ; * + / []	a pound sign: #

Use More than One Security Mechanism

Learn how to set up more than one security mechanism.

If your replication system supports multiple security mechanisms, you may need to install more than one security mechanism on your Replication Server to ensure that both ends of each pathway can support the same mechanism. In this scenario, you can:

1. Configure the Replication Server, for all routes, connections, and other pathways, using **configure replication server**. Make sure that the default security mechanism name is the first one listed under SECURITY in the `libtcl.cfg` file.
2. Configure security for the individual pathways that use a different security mechanism. Make sure that the security mechanism is listed in `libtcl.cfg`.

To find out the security mechanisms and supported security parameters of the Replication Server, use the **admin security_property** command. To find out the security mechanisms and current settings of a particular pathway, use the **admin security_settings** command.

See also

- *View Information About Security Services* on page 294

Manage Replication Server Security

- *Secure Different Types of Network Pathway* on page 280

Potential Security Issue

If different security mechanisms are used at the primary and replicate databases and Adaptive Server user names cannot be guaranteed unique at these sites, a potential security breach exists for request stored procedures.

If this scenario exists on your system, you can make sure that security is maintained by turning “off” the **dsi_exec_proc** parameter for the connection with the primary database. Executing **alter connection** and turning **dsi_exec_proc** “off” disables the Replication Server request-stored-procedures feature.

Here is the syntax:

```
alter connection to data_server.database
set dsi_exec_request_sproc 'off'
```

Manage SSL Security

The Replication Server secure sockets layer (SSL) Advanced Security option provides session-based security. SSL is the standard for securing the transmission of sensitive information, such as credit card numbers and stock trades, over the Internet.

SSL Overview

SSL, also called Transport Layer Security (TLS), provides a lightweight, easy-to-administer security mechanism with several encryption algorithms. It is intended for use over those database connections and routes where increased security is required.

SSL uses certificates issued by certificate authorities (CAs) to establish and verify identities. A certificate is like an electronic passport; it contains all the information necessary to identify an entity, including the public key of the certified entity and the signature of the issuing CA.

See documentation from your third-party SSL security mechanism for instructions for using that software. See also the Internet Engineering Task Force (IETF) Web site for additional information.

An SSL installation requires these items:

- Certificate authority – a valid entity that verifies and signs certificates. Each CA has its own verification policies for issuing digital signatures.
- Certificate – an electronic document that identifies a server, a user, an organization, or other entity. A certificate contains the public key of the certified entity and a signature of the issuing CA.
- Filter – a special network driver that filters information delivered to and from a port.
- Identity file – concatenates a certificate and the certificate’s private key.

- Trusted roots file – contains a list of certificates. Open Client/Server accepts only those CAs listed in the trusted roots file.
- CipherSuites – a set of cryptographic algorithms for authenticating a client and server, transmitting certificates, encrypting data, and establishing security session keys.

The SSL protocol runs above TCP/IP and below application protocols such as HTTP or TDS. Before the SSL connection is established, the server and client exchange a series of I/O round trips to negotiate and agree upon a secure encrypted session. This process is called the SSL handshake.

The SSL Handshake

The standard SSL handshake consists of three steps.

The standard SSL handshake steps are:

1. The client sends a connection request, which includes the SSL options the client supports, to the server.
2. The server returns its certificate and a list of supported encryption algorithms called CipherSuites, key-exchange algorithms, and digital signatures.
3. Both client and server agree on a CipherSuite, and a secure, encrypted session is established.

SSL on Replication Server

Learn how Replication Server supports SSL.

Replication Server does not directly invoke SSL APIs. Replication Server SSL support is based on functionality provided by Sybase Open Client/Server. Sybase uses the SSS Plus™ library API from Certicom to support SSL in Open Client/Server applications. See the *Open Server Server-Library/C Reference Manual* for a complete description of Open Client/Server support for SSL.

Replication Server Advanced Security option supports server authentication and data encryption; it does not support client authentication. For incoming connections, Replication Server supports both SSL and non-SSL ports. For outgoing connections, Replication Server supports both SSL and non-SSL ports on the target server. Clients must log in to the server using a user name and password. Replication Server verifies the user name and password. Once this connection is made, a secure encrypted session can be established.

Use of SSL-secured links can impact Replication Server performance. Sybase recommends SSL only for those connections or routes that transmit sensitive data.

SSL Requirements

There are several requirements for SSL.

- Replication Server 15.0 or later supports TLS version 1.0 or 2.0; it does not support SSL version 3.0.

Manage Replication Server Security

- SSL requires Replication Server version 12.5 and later. Earlier versions of Replication Server do not support SSL.
- The Replication Server Administrator must generate and secure the server certificates and trusted root CA certificates as files outside Replication Server.

Setting Up SSL Security on Replication Server

Learn how to set up SSL services on Replication Server.

Prerequisites

Before setting up SSL services on Replication Server, review the SSL Plus user documentation and documentation for any third-party SSL security software you are using.

Task

1. Edit `$$SYBASE/$$SYBASE_OCS/config/libtcl.cfg` to include SSL driver location.
2. Edit `$$SYBASE/config/trusted.txt` to include trusted CA certificates.
3. Obtain a certificate from a trusted CA for each Replication Server accepting SSL connections.
4. Create the identity file that concatenates a certificate and its private key.
5. Use `rs_init` to enable SSL on Replication Server and to add an encrypted SSL password to the Replication Server configuration file.

Note: You can also enable and disable SSL on Replication Server using `configure replication server` and the `use_ssl` option.

6. Create an SSL entry in the Replication Server interfaces file or directory service.
7. Restart Replication Server.

See *Replication Server Configuration Guide > Secure Sockets Layer* for detailed instructions for each of these steps.

Enable SSL Security on Replication Server

You can enable or disable the SSL security feature using `rs_init`, or you can use `configure replication server` with the `use_ssl` option.

To enable SSL with `use_ssl`, enter:

```
configure replication server
  set use_ssl to 'on'
```

Set `use_ssl` to “off” to disable SSL. By default, SSL is not enabled on Replication Server. When `use_ssl` is “off,” Replication Server does not accept SSL connections.

`use_ssl` is a static option. You must restart Replication Server after you change its value.

Command Auditing

Enable command auditing for Replication Server to record information about user actions and commands.

Types of Actions Recorded by Command Auditing

When you enable command auditing, Replication Server records information on each user action that results in an entry to the audit log:

- Execution of any command belonging to the classes of commands that are recorded.
- Failure to execute a command because the user lacks the required permissions. For example, **create user** requires sa permission.
- Failure to execute a command for other reasons, such as syntax errors.
- User who started Replication Server.
- Number of failed login attempts.

Note: Command auditing records passwords in the log file as asterisks: *****.

Command auditing records entire batches of commands, even if there is only one command that includes the type of information recorded, or that belongs to a command class that is recorded. Command auditing inserts the error description in the log, before the batch.

Classes of Commands Recorded by Command Auditing

Replication Server records all Replication Command Language data definition language (DDL) commands when you enable command auditing, except system information commands, such as the **admin** commands you use only to display information, and that you do not use to configure Replication Server.

For example, Replication Server records **alter route**, **configure replication server**, **connect**, **create connection**, **create partition**, **create user**, **grant**, **ignore loss**, **sysadmin sqm_purge_queue**, and **wait for create standby**, but does not record **admin config** and **admin sqm_readers**.

Command Auditing Configuration

You can configure command auditing to record the commands in the Replication Server log, which is the default destination, or in any other destination file you provide. Use **configure replication server** with **audit_enable** to set auditing on or off:

```
configure replication server
set audit_enable to {on|off}
```

Set **audit_enable** to on to enable command auditing. The default is off.

Optionally set **audit_dest** to specify the destination file for the command log:

Manage Replication Server Security

```
configure replication server
set audit_dest to ['log'|'filename']
```

You can set **audit_dest** if **audit_enable** is on. The default value for **audit_dest** is **log**, which is the Replication Server log. Specify a file name and path for any other destination. Ensure that the owner has read and write permissions for any file you specify in **audit_dest**. In UNIX, Replication Server creates the log file with 0600 permissions if the file does not already exist. If you create your own log file in UNIX with different permissions such as 0666, Replication Server retains your permissions. For example, to log commands to /tmp/RSaudit.log:

```
configure replication server
set audit_dest to '/tmp/RSaudit.log'
```

Example Messages in the Command Auditing Log

In the log file, "AUDIT" precedes the corresponding command or user action.

- **Command that executes successfully:**

```
I. 2012/03/29 02:30:23. AUDIT: incoming command (issued by sa):
sysadmin site_version
```
- **Command that fails to execute because the user lacks the required permissions:**

```
I. 2012/03/29 02:31:46. AUDIT Command failed: SA permission
required for:
I. 2012/03/29 02:31:46. AUDIT: incoming command (issued by user1):
shutdown
```
- **Command that fails to execute for other reasons:**

```
I. 2012/03/29 03:18:15. AUDIT The following command batch has one
or more failures:
I. 2012/03/29 03:18:15. AUDIT: incoming command (issued by sa):
sysadmin badcommand
```
- **User who started Replication Server on a UNIX platform –**

```
I. 2012/03/29 03:18:03. AUDIT: incoming command (issued by none):
Repserver started by username: ny_admin1
```
- **Failed login attempt to Replication Server –**

```
I. 2012/03/22 02:12:52. AUDIT: Failed login attempt for user sa
```
- **A batch of commands that executes successfully:**

```
I. 2012/03/29 03:22:19. AUDIT: incoming command (issued by sa):
create user user3 set password *****
I. 2012/03/29 03:22:19. AUDIT: incoming command (issued by sa):
sysadmin site_version
```
- **A batch of commands with one command that fails to execute because the user lacks the required permissions:**

```
I. 2012/03/29 03:23:14. AUDIT Command failed: SA permission
required for:
I. 2012/03/29 03:23:14. AUDIT: incoming command (issued by user1):
admin who
I. 2012/03/29 03:23:14. AUDIT: incoming command (issued by user1):
shutdown
```
- **A batch of commands with at least one command that fails to execute for other reasons:**

```
I. 2012/03/29 03:24:08. AUDIT The following command batch has one
or more failures:
```

```
I. 2012/03/29 03:24:08. AUDIT: incoming command (issued by sa):
sysadmin site_version
I. 2012/03/29 03:24:08. AUDIT: incoming command (issued by sa):
sysadmin badcommand
```

Security Recommendations

Recommendations for Replication Server security issues such as for performing administrative tasks, SSL, encryption, permissions and roles, and the configuration file..

- As a best practice, perform administration tasks only on the local Replication Server host. By default, Replication Server does not prevent an administrator who knows the Replication Server host name and port number, from accessing and administering the Replication Server remotely.
- Wait for a master database transaction such as creating a new user or changing a password, to replicate successfully to all replicate Adaptive Servers before executing a user database transaction such as creating a table, that depends on the master database transaction. Replication Server maintains the transaction commit order for transactions executed within a single Adaptive Server database. However, Replication Server does not maintain such an order for transactions executed across multiple Adaptive Server databases. For example, at the primary Adaptive Server:

- To create a master database transaction such as creating the mylogin user , use the sa user to enter:

```
sp_addlogin 'mylogin', 'password'
go
use mydb
go
sp_adduser
'mylogin'
go
```

- To create a user database transaction such as creating the mytab table with the mylogin user ID, enter:

```
use mydb
go
create table mytab (mycol int)
go
```

It is possible for Replication Server to replicate the **create table** command before **sp_addlogin** procedure which causes the **create table** to fail on the replicate Adaptive Server because the mylogin user does not yet exist at the replicate database.

- Replication Server can use Secure Sockets Layer (SSL) to provide session-based security. SSL uses certificates issued by certificate authorities (CAs) to establish and verify identities.

If a SSL certificate is compromised, you must request for a new certificate from the CA with a new Replication Server name and certificate number.

- The administrator should control permissions on the Replication Server log to provide monitor-only access to auditors. By default, any user that you create in Replication Server,

Manage Replication Server Security

who has not been granted any roles, has monitor-only access to RSSD tables sufficient for a support role.

- Consider disk-level encryption for sensitive data in stable queues.
Even with connectivity based on SSL between the primary and replicate databases and Replication Server, Replication Server must persist data temporarily in the stable queues, and this persisted data is not encrypted.
- Sybase recommends that you use SSL for connections or routes that transmit sensitive data. The Replication Server Secure Sockets Layer (SSL) Advanced Security option provides session-based security.
- Replication Server stores initial configuration properties such as host name, port, user name, and password, in a file with the `.res` suffix that the `rs_init` utility uses. Set the appropriate umask permissions in UNIX or directory permission in Windows for the `.res` file, or delete the file if you do not require it.

Although `rs_init` does not require the `.res` file after the initial configuration, Replication Server stores the file in the operating system file system protected only by the operating system permissions.

See also

- *Manage SSL Security* on page 296

Manage Replicated Tables

Learn the preparations, the procedures, and the specific commands used to set up manage replicate tables, table replication definitions, and publications.

You can copy data from one database to another in different ways depending on which method best suits the needs at your site:

- Using a single database replication definition that lets you choose whether or not to replicate individual tables, transactions, functions, system stored procedures, and data definition language (DDL). You can use multisite availability (MSA) for database replication.
- Using function replication definitions, where each one identifies a specific system stored procedure for replication.
- Using table replication definitions, where each one identifies a specific table for replication and, optionally, specifies a subset of columns to be replicated.

See also

- *Manage Replicated Objects Using Multisite Availability* on page 449
- *Manage Replicated Functions* on page 379

Introduction to Managing Replicated Tables

Replication Server allows you to copy and update data from a table in one database—the primary—to a table in another database—the replicate.

Note: The primary database is also referred to as the “source.” The replicate database is also referred to as the “destination.”

To establish a table as the source, you create a replication definition that specifies the location of the data you want to copy and describes the structure of the table in which the data resides.

Before you copy data from the source table, you must also create a duplicate of the table in the destination data server. Then, in the Replication Server that manages the destination table, you create a subscription to the replication definition. A subscription resembles a SQL **select** statement.

If you do not want to duplicate all of a table’s data, Replication Server lets you specify a subset of columns to copy in the replication definition or use a **where** clause in the subscription to specify a subset of rows to receive.

You can include replication definitions for related tables and stored procedures in a publication and then create subscriptions against all of them as a group. When you use

Manage Replicated Tables

publications you can organize your subscriptions and monitor status information for all subscriptions in the group with a single command.

You can change the datatype of replicated values using the heterogeneous datatype support (HDS) feature. HDS allows you to translate the datatype of a replicated column value to a datatype acceptable to the replicate data server. You can use HDS in Sybase environments, in non-Sybase environments, and in mixed Sybase and non-Sybase data server environments.

See also

- *Manage Subscriptions* on page 397
- *Subscription Example* on page 428

Plan a Replication System

Planning a replication system includes considering the design, being aware of the restrictions on data replication, and preparing the replication system.

Design Considerations

There are several design considerations when you set a replication system.

When you set up a replication system, consider the following:

- Security, including user login names and passwords, permissions required for executing commands, and third-party security systems.
- Concurrency control; specifically, protecting your replication system from conflicts that may result from data being modified by one client when it is also being used by another. See *Replication Server Design Guide > Introduction > Transaction Management*.
- CPU, memory, disk, and network resources. See *Replication Server Design Guide > Capacity Planning*.
- Your replicated data model and routing scheme.
- Requirements for using heterogeneous data servers as data sources or data destinations. See *Replication Server Design Guide > Non-ASE Data Server Support*.
- Compatibility between Adaptive Servers and Replication Servers of different versions and restrictions on data replication.

For information about Sybase compatibility issues, see the release bulletin for your platform.

See also

- *Manage Replication Server Security* on page 231
- *Replication Server and Distributed Database Systems* on page 7
- *Routing Schemes* on page 167
- *Mixed-Version Replication Systems* on page 17

Restrictions on Data Replication

There are several restrictions when you design a replication system.

When you design your replication system, you should also consider these restrictions:

- Adaptive Server and Replication Server system tables cannot be copied during normal replication. However, the execution of supported commands and system procedures on certain system tables can be copied in warm standby applications. See *Replicated Information for Warm Standby* in the *Replication Server Administration Guide Volume 2*. In addition, some data is automatically copied between RSSDs in the replication system.
- Generally tables that you want to copy must have unique primary keys. With certain limitations, Replication Server supports the replication of a table to Adaptive Server even if the table does not have a unique key.
- Client applications should not update unique index or primary key columns in a way that a key could duplicate the key of another row. Because of the way Replication Server copies transactions, this type of update could result in duplicate rows or errors at replicate databases.

For example, if `pk_col` is the primary key column for `table1`, the following command could cause errors or incorrect data at the replicate database:

```
update table1
set pk_col = pk_col + 1
```

If there is a primary key or unique index constraint on the replicate table, the updates fail and the DSI thread for the replicate database is suspended.

- If a trigger is associated with a replicate table, do not put a commit statement inside the trigger. Triggers that contain commit statements at replicate sites may cause a duplicate key and make Replication Server recovery fail.
- Replication Servers of different versions can work together in the same replication system, but certain features may be restricted.
- Virtual computed columns cannot be replicated since Replication Agent cannot forward virtual columns to Replication Server, and Replication Server cannot insert or update virtual columns.
- When replicating encrypted columns which have:
 - Domain rules or check constraints defined on the columns, Replication Server bypasses the domain rules or check constraints for insert and update statements.
 - Access rules defined on the columns, Replication Server returns the 2929 error—“The access rule cannot be attached”—when processing **update**, **delete**, or **select** statements.

See also

- *Mixed-Version Replication Systems* on page 17
- *Replication of Tables Without Unique Keys* on page 311

Preparing a Replication System

There are several tasks to complete before you can replicate data.

1. Set up the replication system
 - a) Install Replication Servers. See the Replication Server installation and configuration guides for your platform.
 - b) Create the databases that will be the primary and replicate. See the *Adaptive Server Enterprise Reference Manual* or the documentation for your non-Sybase database software.
 - c) Establish connections from Replication Servers to the primary and replicate databases. See the *Replication Server Configuration Guide* for your platform.
 - d) Establish all necessary routes between Replication Servers.
 - e) Configure and start up Replication Agents for source databases.
2. Verify that all replication system components are working. See *Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server*.

See also

- *Create Database Connections* on page 194
- *Manage Routes* on page 165
- *Manage a Replication System* on page 83

Summary of Procedures for Replicating Tables

The replication procedure consists of several steps required to replicate data between tables using table replication definitions and subscriptions.

1. Plan your replication system and verify that you have prepared the replication system correctly.
2. Create the table as the Database Owner in the primary database, if it does not already exist, or, if there is a different table owner, specify the table owner name when you create the replication definition.
 - In Adaptive Server, use **create table** to create the table, or use **sp_help** to verify that the table exists.
 - If you are replicating data from a source other than Adaptive Server, create the table according to the instructions for your database software. Other data server steps in this procedure may vary for heterogeneous replication.
3. If your database supports stored procedures, you can execute **rs_send_repserver_cmd** in the primary database to create one or more replication definitions for the table from which you want to copy data. Otherwise, in the primary Replication Server, create one or more replication definitions. Each replication definition can be subscribed to by a different site that uses a different table view.

When you create replication definitions, anticipate the requirements for the subscribing table, as described in step 8. The replication definition may contain all or a subset of the columns in the source table. It may specify the same or different table names, owner names, column names, or datatypes for the source and destination tables. It may change the datatype of the replicated value.

4. If you are using publications, execute the following steps at the primary Replication Server.
 - a) Create one or more publications for the tables you want to replicate using **create publication**.
 - b) Create one or more *articles*, replication definition extensions, for each replication definition you want to include in the publication using **create article**. You can include a **where** clause to specify a subset of rows to send to the destination database.
 - c) Validate the publications, using **validate publication**, so that you can create subscriptions against them.
5. Mark the primary table for replication.

In the primary Adaptive Server, use **sp_setreptable** to enable table replication. This step allows the RepAgent thread to forward transactions for the table to the primary Replication Server.

Note: For non-Adaptive Server primaries, see your Replication Agent documentation for instructions on marking tables and columns.

6. If the primary table contains `text`, `unitext`, `image`, or `rawobject` columns, you may need to use **sp_setrepcol** in the primary Adaptive Server to adjust the replication status for these columns.

Note: For non-Adaptive Server primaries, see your Replication Agent documentation for instructions.

7. Prepare a login name for the user creating the subscription. Login names that create subscriptions at replicate Replication Servers must also exist at the primary Replication Server.
8. In the replicate database, create a table that matches the schema published by the replication definition. Create the replicate table as the Database Owner or as the same table owner specified in the replication definition.

In Adaptive Server, use **create table** to create the table, or use **sp_help** to verify that the table exists.

The replicate table may have the same or different name and/or the same or different owner name as the primary table. It may contain all or a subset of the columns in the primary table, with the same or different column names or datatypes. The replication definition must specify any such differences between the primary and replicate tables.

Note: The replicate table may include a column that is not in the replication definition if the column accepts null values, has a defined default value, or you use a custom function string to apply a value to that column.

9. Grant the replicate database maintenance user login name **select**, **insert**, **delete**, and **update** permissions on the replicate table. The maintenance user executes commands for replicated transactions.

10. If necessary, customize your database operations using functions, function strings, and function-string classes. Replication Server function strings execute data server operations.

See *Replication Server Administration Guide Volume 2 > Customizing Database Operations* for details.

11. Create a subscription in the replicate Replication Server. If you are using publications, proceed to step 12.

Log in to a replicate Replication Server and create one or more subscriptions to the table replication definition for the data you want to copy. You can subscribe to all the rows in the replication definition's columns, or use a **where** clause to copy only certain rows.

A replicate database can subscribe to multiple replication definitions of a primary table, but a replicate table can subscribe to only one replication definition of a primary table.

When you create a subscription, the replicate table is filled in with the initial table data in a process called materialization. In most cases, Replication Server copies data into the replicate table automatically. You can also manually materialize the data.

12. If you are using publications, create a publication subscription against the publications created in step 4. Execute **create subscription** at the replicate Replication Server.

When you create a publication subscription, Replication Server creates subscriptions against each article in the publication. Article subscriptions do not contain **where** clauses.

13. Check the subscription status.

Verify that the subscription data has fully materialized in the replicate database and that transactions are replicating successfully.

See also

- *Specify Data for Replication* on page 31
- *Plan a Replication System* on page 304
- *Use the create replication definition Command* on page 315
- *Create Multiple Replication Definitions Per Table* on page 327
- *Use Publications* on page 364
- *Mark Tables for Replication* on page 329
- *Replicate text, unitext, image, and rawobject Columns* on page 336
- *Manage Subscriptions* on page 397
- *Manage Replication Server Security* on page 231

- *Publication Subscriptions* on page 438
- *Commands for Managing Function Replication Definitions* on page 382
- *Subscription Commands* on page 416
- *Subscription Example* on page 428
- *Preparing a Replication System* on page 306

Commands for Managing Table Replication Definitions

Replication Server provides commands and stored procedures to manage table replication definitions.

Table 25. Commands for Managing Table Replication Definitions

Command	Task
create replication definition	Creates a replication definition for a primary table, which describes the columns you want to copy, the location of the table, and other information.
alter replication definition	Modifies an existing replication definition in a variety of ways, including: <ul style="list-style-type: none"> • Adding or dropping columns • Add or drop primary keys • Add or drop searchable columns from the replication definition • Specifying different replicate table, table owner, column name, or datatype • Changing minimal column replication • Add or alter column-level datatype translations • Change <code>text</code>, <code>unitext</code>, <code>image</code>, or <code>rawobject</code> column replication status • Change how the replication definition is used in replicating to a standby database
drop replication definition	Removes a replication definition from the replication system. You must drop all subscriptions for a replication definition before you can drop the replication definition.
rs_send_repserver_cmd	Executes replication definition change requests directly at the primary database. You can use the <code>rs_send_repserver_cmd</code> stored procedure to execute the create replication definition , alter replication definition , and drop replication definition commands at the primary database.
admin verify_repserver_cmd	Verifies that Replication Server can successfully execute a replication definition change request.

See also

- *Create Replication Definitions* on page 310
- *Alter Replication Definitions* on page 349
- *Drop Replication Definitions* on page 362
- *Execute Replication Definition Changes Directly at the Primary Database* on page 351
- *Verify Replication Definition RCL Commands* on page 353
- *Using the Replication Definition Change Request Process* on page 354

Create Replication Definitions

A replication definition describes the primary table to Replication Server, specifying the columns you want to copy.

It may also describe attributes of the replicate table. Replicate tables that match the specified characteristics can subscribe to the replication definition. You can create multiple replication definitions for the same primary table, each customized for a particular use.

To create a replication definition, you can execute one of the following:

- **create replication definition** at the Replication Server managing the source table.
- **rs_send_repserver_cmd** with the **create replication definition** clause at the primary database when execute replication definition change requests directly at the primary database.

Information about each replication definition is sent to each qualifying Replication Server with a route from the primary Replication Server. There are restrictions in mixed-version systems.

Replication definitions are stored in the `rs_objects` and `rs_columns` system tables in the RSSD. The primary version of a replication definition resides at the primary Replication Server.

See also

- *Create Multiple Replication Definitions Per Table* on page 327
- *Use the create replication definition Command* on page 315
- *Execute Replication Definition Changes Directly at the Primary Database* on page 351
- *Replication Definition Restrictions in Mixed-Version Systems* on page 329

Replication Definition Settings

There are several pieces of information you require to create a replication definition.

Each replication definition must include the following information:

- The name of the replication definition.

- The names of the primary and replicate tables.
Replication Server assumes that the replication definition name is the name of both the source and destination tables, unless you specify differently.
To use table names containing spaces or special characters, enable quoted identifiers support.
- The name of the data server and database where the primary table is located.
- The column names and datatypes that you want to copy. You can copy all or a subset of the primary table's columns. The replicate column names and datatypes are the same as the primary column names and datatypes, unless you specify differently.
To use column names containing spaces or special characters, enable quoted identifiers support.
- The primary key—one or more columns that uniquely identify each row in the source table.
A primary key cannot include encrypted columns, or columns with `text`, `unitext`, `image`, `rawobject`, or `rawobject` in-row datatypes. See *Datatypes* in the *Replication Server Reference Manual* for datatypes that Replication Server supports.

Optionally, a replication definition may also include:

- The names of the owners of the primary and replicate tables. The default table owner is the Database Owner (dbo).
- The names of searchable columns—columns that can be specified in the **where** clause of a subscription to indicate the rows from the primary table to copy into the replicate table.
- For a warm standby application, whether to use the replication definition to copy data into a standby database and whether to copy all of the table's columns or just the columns in the definition's column list.
- Whether to copy only the minimal number of columns required for update and delete operations. This option may enhance overall system performance.
- Replication options for `text`, `unitext`, `image`, and `rawobject` columns.
- Column-level datatype translations.

Replication of Tables Without Unique Keys

With certain limitations, Replication Server supports the replication of a table to Adaptive Server even if the table does not have a unique key.

If a table does not have a unique key, it is possible that there are two or more rows with the same values. However, DSI shuts down if it cannot find a unique row to apply an operation. The **dsi_top1_enable** parameter instructs DSI to select and update only the first instance of multiple similar rows by setting *unsigned_integer* to 1 in the **top unsigned_integer** clause of the Adaptive Server **select** statement.

Set the **dsi_top1_enable** parameter to on to support replication of tables without unique keys. To set the parameter for:

- A specific connection to a replicate Adaptive Server database, enter:

Manage Replicated Tables

```
alter connection to data_server.database set dsi_top1_enable to {on | off}
```

- All replicate database connections to the Replication Server, enter:

```
configure replication server set dsi_top1_enable to {on | off}
```

The default is off.

Do not use **configure replication server** to set **dsi_top1_enable** to on if there are connections to non-Adaptive Server replicate databases. Instead, use **alter connection** to set the parameter individually for each Adaptive Server replicate database connection.

Limitations

Replication Server does not support replication of a table without unique keys if:

- There is a Java object datatype in the table. See *Java Datatypes* in the *Replication Server Reference Manual*.
- There is a customized function string for the table.
- There is a table replication definition for the table.
- Transactions use updates and deletes with the **like** option in the **where** clause at the primary database.
- The replicate database is not Adaptive Server.

Quoted Identifiers

You can enable quoted identifier support for object names that need to be enclosed in double quote characters to be parsed correctly.

Quoted identifiers are object names that contain special characters such as spaces and nonalphanumeric characters, start with a character other than an alphabetic character, or that correspond to a reserved word, and need to be enclosed in double quote characters to be parsed correctly.

Enable quoted identifier support for the DSI for table and column names because the object names:

- contain special characters such as spaces and nonalphanumeric characters
- start with a character other than an alphabet
- correspond to a reserved word

To ensure that replication proceeds correctly for objects with quoted identifiers created in versions of Adaptive Server earlier than 15.7, update the Adaptive Server system catalog after you upgrade Adaptive Server, but before you restore replication. See *Updating the Adaptive Server System Catalog to Support Quoted Identifiers* in the *Replication Server Configuration Guide*.

Enable Quoted Identifiers Support

Use **create connection** or **alter connection** with **dsi_quoted_identifier** to enable quoted identifier support.

See the *Parameters Affecting Database Connections* table in **alter connection** in the *Replication Server Reference Manual* for the full command description.

Warning! If you set **dsi_quoted_identifier** to **on** for an Adaptive Server database connection, you must not include double quotes in any stored procedure you want to replicate. Otherwise, the DSI thread shuts down.

Data servers receive quoted identifiers differently. Adaptive Server and Microsoft SQL Server need a special connection set-up for quoted identifiers because they do not expect quoted identifiers. Oracle and UDB do not need a special connection to accept quoted identifiers.

Use **admin config** to check the value of **dsi_quoted_identifier**.

See also

- *Configuration Parameters Affecting Physical Database Connections* on page 201

System Function to Support Quoted Identifiers

The **rs_set_quoted_identifier** function string is added to set the DSI connection to Adaptive Server and Microsoft SQL Server.

When **dsi_quoted_identifier** is set to **on**, Replication Server sends the **rs_set_quoted_identifier** function string to the replicate database as part of the connection configuration when the connection is established. Replication Server uses this function string to allow quoted identifiers to be sent through the connection.

See *Replication Server Reference Manual > Replication Server System Functions > rs_set_quoted_identifier* and *Replication Server Administration Guide Volume 2 > Customize Database Operations > Working with Functions, Function Strings, and Classes > Summary of System Functions*.

Mark Identifiers as Quoted

Use the **create replication definition** or **alter replication definition** commands with the **quoted** parameter to mark identifiers as quoted identifiers.

When **dsi_quoted_identifier** is on for a replicate server connection and when **dsi_quoted_identifier** is subscribed to a replication definition that has identifiers marked as quoted identifiers, the marked identifiers are enclosed in double quotes.

When you set **dsi_quoted_identifier** to on and mark an identifier, the replicate database that subscribes to the replication definition receives the marked identifier as a quoted identifier. If **dsi_quoted_identifier** is off, the markings are ignored and the identifier sent to the replicate database is not enclosed in double quotes.

Note: When replicating to a warm standby database and to replication definition subscribers, and the primary table name is marked as quoted but the replicate table name is not, or vice-versa, Replication Server sends both the primary table name and the replicate table name as quoted.

Example 1

To create the **authors_repdef** table replication definition for the **authors** table with the **authors_col_1** column as a quoted identifier:

```
create replication definition authors_repdef
  with primary at primaryDS.primaryDB
  with all tables named "authors"
  ("authors_col_1" int quoted, "authors_col_2" int)
  primary key ("authors_col1")
```

Example 2

To mark as quoted the **authors** table:

```
alter replication definition authors_repdef
with replicate table named "authors" quoted
```

Example 3

To unmark the column **authors_col_1**:

```
alter replication definition repdef
with replicate table named "authors"
alter columns with "authors_col_1" not quoted
```

See **create replication definition** and **alter replication definition** in the *Replication Server Reference Manual*.

Quoted Identifiers Without Replication Definition Support

Replication Server can add quotes to all identifiers without checking whether the identifier is quoted in a replication definition or in the primary Adaptive Server database.

Setting **dsi_quoted_identifier** to **on** enables quoted identifier support for primary tables only created in Adaptive Server 15.7.1 and later.

Setting **dsi_quoted_identifier** to **always** automatically adds quotes to object names and ensures that replication continues for all primary data servers supported by Replication Server. Otherwise, the DSI connection to the replicate database can shut down and stop replication if Replication Server processes table names that are not quoted and not specified in a table replication definition with the **quoted** clause. See the *Replication Server Release Bulletin* for supported Adaptive Server versions and the *Replication Server Options Release Bulletin* for supported non-Adaptive Server data servers and versions.

You can use the **always** option for a specific table or for all tables when you configure **dsi_quoted_identifier** for a connection with **create connection** or **alter connection**.

Example

To ensure that replication proceeds even if some table names are not quoted in the corresponding table replication definitions, set the **always** option for all tables for a new connection to the `rdb` replicate database in the `NY_DS` data server. At the replicate Replication Server, enter:

```
create connection to NY_DS.rdb
...
set dsi_quoted_identifier to always
...
go
```

Mixed-Version Restrictions

The quoted identifier feature is not supported in mixed version environments. For replication of a quoted identifier to succeed, the primary Replication Server and the Replication Server that connects to the replicate data server version must be version 15.2 and later. However, intermediate Replication Servers in a route can be of lower versions.

Use the create replication definition Command

Use **create replication definition** to describe to Replication Server the characteristics of a table you want to replicate.

Execute **create replication definition** at the Replication Server that manages the source table database, or you can use the **rs_send_repserver_cmd** stored procedure with the **create replication definition** clause to execute replication definition change requests directly at the primary database.

A replication definition must include the name of the source data server and database.

This example shows how to create a basic replication definition named **publishers** for source and destination tables with the same name. The primary database is `pubs2` managed by the `TOKYO_DS` data server. All of the table columns are included and the `pub_id` column is specified as the primary key.

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
(pub_id char(4), pub_name varchar(40),
city varchar(20), state char(2))
primary key (pub_id)
```

See *Replication Server Reference Manual > Replication Server Commands > create replication definition* for complete command syntax and usage guidelines.

See also

- *Execute Replication Definition Changes Directly at the Primary Database* on page 351

Specify the Replication Definition Name and Table Names

A replication definition has a global name space—that is, at every Replication Server with routes from the primary Replication Server, the name refers to the same replication definition.

Replication Server cannot always enforce the unique-name requirement when you enter **create replication definition**. You must ensure that there is no existing replication definition (table or function) with the same name when you create a new replication definition.

By default, the replication definition name is the name of *both* the source and destination tables.

In some instances, you may need to use different names for your source and destination tables, or different names for your tables and replication definitions. Include one of the optional clauses **with all tables named**, **with primary table named**, or **with replicate table named** to specify table names where they differ from the replication definition name.

When Source and Destination Tables Share the Same Name

When the source table and all destination tables share the same name but you want to give the replication definition a different name, use **with all tables named** to specify the table names.

For example, to create a replication definition named **publishers_rep** for source and destination tables named `publishers`, enter this command:

```
create replication definition publishers_rep
with primary at TOKYO_DS.pubs2
with all tables named publishers
...
```

When Source and Destination Tables Have Different Names

When the source table and any destination tables have different names, use **with primary table named** to specify the name of the source table, or use **with replicate table named** to specify the destination table name.

You can use one of these clauses or both of them together.

If you do not specify different table names, the replication definition name is assumed by Replication Server to be the name of *both* the source and destination tables.

For example, to create a replication definition named **publishers_rep** for a source table named `publishers1` and destination tables named `publishers2`, enter:

```
create replication definition publishers_rep
with primary at TOKYO_DS.pubs2
with primary table named publishers1
with replicate table named publishers2
...
```

For a replication definition and a source table named `publishers`, and destination tables named `publishers2`, enter:

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
with replicate table named publishers2
...
```

In this example, the **publishers** replication definition also becomes the source table's name.

Specify the Name of the Source or Destination Table Owner

You can specify the table owner's name as an optional qualifier along with the name of the source or destination table. Data server operations may fail if the table owner does not correspond to what is specified in the replication definition.

For example, to create a replication definition for the `publishers` source table and the `publishers2` destination table owned by the user "ravi," enter:

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
with replicate table named ravi.publishers2
...
```

Specify Column Names and Datatypes

When you create a replication definition, list the names and datatypes of the columns from the table that you want to copy.

A column's name and datatype will be the same in the replicate table as in the primary table unless you specify a different replicate (published) column name or datatype.

Enclose the names of all of the columns and their datatypes in parentheses. For multiple columns, separate each column and its datatype from the next column with a comma.

For example, the following command creates a replication definition named **publishers_rep1** for source and destination tables named `publishers`. It includes all the columns and their datatypes.

```
create replication definition publishers_rep1
with primary at TOKYO_DS.pubs2
with all tables named publishers
(pub_id char(4),
pub_name varchar(40),
city varchar(20),
state char(2))
primary key (pub_id)
```

The following command creates a replication definition named **publishers_rep2** that omits the `city` column. Destination sites that do not require this column can subscribe to this replication definition.

```
create replication definition publishers_rep2
with primary at TOKYO_DS.pubs2
with all tables named publishers
(pub_id char(4),
pub_name varchar(40),
```

Manage Replicated Tables

```
state char(2))
primary key (pub_id)
```

Performance is best if columns are listed in the same order in the replication definition as in the tables themselves.

You can use only native and user defined datatypes supported by Replication Server. If a primary table has columns with user-defined datatypes, you must use a compatible supported datatype in the replication definition. You can also employ user-defined datatypes supplied with Replication Server as part of the installation process.

See *Replication Server Reference Manual > Topics > Datatypes* for complete details on the datatypes supported by Replication Server.

When Source and Destination Columns Have Different Names

When you want only one replication definition for a source table, and the source column names differ from their destination counterparts, use the *column_name as replicate_column_name* clause in the replication definition.

For example, for a source table named `publishers1` and a destination table named `publishers2`, where the source column `pub1_name` corresponds to the destination column `pub2_name`, enter this:

```
create replication definition publishers_rep
with primary at TOKYO_DS.pubs2
with primary table named publishers1
with replicate table named publishers2
(pub_id char(4),
pub1_name as pub2_name varchar(40),
city varchar(20),
state char(2))
primary key (pub_id)
```

Datatypes in Multiple Primary Table Replication Definitions

When you create multiple replication definitions for the same source table, the declared column datatype (the column datatype in the primary table) must be the same, except when the column's datatype is `rawobject` or `rawobject in row`, which correspond respectively to the `image` and `varbinary` datatypes.

Specifically you can:

- Declare a column's datatype as `rawobject` in one replication definition, but declare the same column's datatype as `image` in another replication definition for the same table
- Declare a column's datatype as `rawobject in row` in one replication definition, but declare the same column's datatype as `varbinary` in another replication definition for the same table

The `replicate` (published) column datatype can be different between replication definitions for the same table, with no restrictions.

When a column is listed in an existing replication definition for a primary table, specifying the column datatype is optional in subsequent replication definitions for the same primary table—the datatype is inherited from the previous replication definition and retained for the subsequent definition, even if the first definition (where you specified the datatype) is dropped.

To change a column datatype, use the **alter replication definition** command.

See also

- *Alter Column Datatypes* on page 360

Additional Columns in the Replicate Table

The replicate table may include a column that is not in the replication definition if the column has a defined default value or you use a custom function string to apply a value to that column.

Columns can be specified to accept null values in **create table**. When source rows are copied to the destination table, extra columns are filled with null values or may be updated separately by the local data server.

Include text, unitext, image, and java Columns

To copy `text`, `unitext`, `image`, or the Java datatypes `rawobject` and `rawobject in row` column data to any destination site, include those columns in the replication definition.

Replicating `text`, `unitext`, `image`, or Java columns involves additional special procedures and considerations.

See also

- *Replicate text, unitext, image, and rawobject Columns* on page 336
- *Java Datatypes in Replication Server* on page 333

Use Special Datatypes

To distribute updates to particular sites, use the `rs_address` special datatype with bitmap subscriptions.

You can use the `identity` special datatype if the table you are copying contains an `identity` column.

You can also use the `timestamp` special datatype if the table you are copying contains a `timestamp` column.

See also

- *Bitmap Subscriptions* on page 433
- *Replicate identity Columns* on page 347
- *Replicate timestamp Columns* on page 347
- *Using the rs_address Datatype* on page 346

Use User-defined Datatypes

To change the datatype of the replicated value at the primary database to a datatype acceptable to the replicate database, use the **map to** clause of the **create replication definition** command.

See also

- *Translate Datatypes Using HDS* on page 372

Specify the Primary key

The primary key is the column or combination of columns that uniquely identifies each row.

Although many data servers, including Adaptive Server, allow tables that contain duplicate rows, Replication Server requires that the source and destination tables have unique values for the primary key columns in each row.

You must include the **primary key** clause in **create replication definition** to identify the primary key columns in the source table. Primary key columns must also be included in the column list.

When Replication Server applies the default **rs_update** or **rs_delete** function string at a destination site, it specifies values for the primary key in the **where** clause of the update or delete statement.

Enclose the names of the primary key columns in parentheses. For example:

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
(pub_id char(4), pub_name varchar(40),
city varchar(20), state char(2))
primary key (pub_id)
```

For multiple primary key columns, separate each column from the next with a comma.

Note: You cannot include columns of text, unitext, image, rawobject, rawobject in row, or rs_address datatypes as part of the primary key.

Specify Searchable Columns

Use **searchable columns** in **create replication definition** to specify which columns to use in the **where** clause of **create subscription** or **define subscription** (or **create article** for publications) to restrict the rows copied to a subscribing site.

If you do not include a **searchable columns** clause in a replication definition, you cannot use a **where** clause in a subscription or article that references that replication definition.

Enclose the names of the searchable columns in parentheses. For multiple searchable columns, separate each column from the next with a comma.

In the following example, three columns, pub_id, pub_name, and state, are specified as searchable columns. You can include any of these columns in a subscription's **where** clause.

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
```

```
(pub_id char(4), pub_name varchar(40),
city varchar(20), state char(2))
primary key (pub_id)
searchable columns (pub_id, pub_name, state)
```

See also

- *Use the where Clause* on page 418

Restrictions on Searchable Columns

Searchable columns have several restrictions.

Searchable columns have these restrictions:

- You cannot specify `text`, `unitext`, `image`, or `Java rawobject` or `rawobject` in row columns as searchable columns.
- Columns included in the **searchable columns** clause cannot have null values.
- To perform bitmap comparison using the **where** clause in the subscription, you must include any columns that use the `rs_address` datatype in the replication definition's **searchable columns** clause.
- The more searchable columns in the **searchable columns** list of a replication definition, the slower Replication Server processes subscriptions; that is, the fewer searchable columns, the more efficiently Replication Server evaluates rows against subscriptions for the table.

See also

- *Using the rs_address Datatype* on page 346

Replicate the Minimal Set of Columns

To enhance replication system performance, specify **replicate minimal columns** in **create replication definition**. This clause lets you send only those columns that are required for delete and update operations to replicate databases.

Normally, Replication Server sends all the columns in each row when applying updates and deletes, as well as inserts, in each replicate database. Replication Server normally sends maximum columns to the standby database—if replication definitions are not used for the table or the replication definitions are not used for the standby connection.

Note: You must send all columns when replicating to SQL Remote databases. Do not send minimal columns or replication will fail.

When you set **replicate minimal columns**:

- For a **delete** operation, the source Replication Server sends only the primary key columns to destination Replication Servers or the standby database.

Manage Replicated Tables

- For an **update** operation, the source Replication Server sends only the columns modified by the update operation and the primary key columns, to destination Replication Servers or the standby database.

Note: replicate minimal columns does not apply to insert operations, for which all columns are copied.

A destination Replication Server uses the primary key columns in constructing the data server commands that it applies to the replicate or the standby database.

The following replication definition includes **replicate minimal columns**:

```
create replication definition publishers
with primary at TOKYO_DS.pubs2
(pub_id char(4), pub_name varchar(40),
city varchar(20), state char(2))
primary key (pub_id)
replicate minimal columns
```

Change Minimal Columns Setting

Use **alter replication definition** to change an existing replication definition to replicate only the minimal set of columns or to replicate all columns.

Minimal Columns and rs_update and rs_delete Function Strings

If you specify **replicate minimal columns** and need to create non-default **rs_update** and **rs_delete** function strings, use the *rs_default_fs* function string variable to represent the default function string behavior.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Use the Default System Variable* for details.

Minimal Columns and Autocorrection

If you specify **replicate minimal columns**, you cannot also specify autocorrection, which corrects discrepancies that may occur during materialization by converting each **update** or **insert** operation into a **delete** followed by an **insert**.

If you set autocorrection on before you specify minimal columns (for example, using **alter replication definition**), autocorrection is not performed. Replication Server logs informational messages for any update operations.

You must set autocorrection on when you create a subscription using nonatomic materialization. If minimal column replication is set for the replication definition and you create a new subscription that uses nonatomic materialization or the bulk materialization method that simulates nonatomic materialization, autocorrection cannot resolve inconsistencies.

See also

- *Autocorrection for Nonatomic Materialization* on page 401
- *Subscription Materialization Methods* on page 398

Use Replication Definitions with Warm Standby Applications

You do not need to use replication definitions with warm standby applications. However, you can use them to control the flow of information to the standby database—even though no subscriptions are needed.

You can create replication definitions just for replication to the standby database or use existing replication definitions for this purpose.

Use **send standby** in **create replication definition** as follows:

- Use **send standby** in any form to replicate transaction data into the standby database using this replication definition. Replication Server uses the replication definition's primary key and minimal columns setting.
- Use **send standby** or **send standby all columns** to send all columns in the table to a standby database.
- Use **send standby replication definition columns** to send only the columns specified in the replication definition to a standby database.

If you omit **send standby**, another replication definition may be used in replicating data for this table to the standby database, or no replication definition may be used.

The replication definition in the following example replicates transactions to a standby database. The primary key and minimal set of columns settings will be used in standby replication. Only the columns specified in the replication definition will be replicated into the standby database—the `city` column is omitted from this replication definition.

```
create replication definition publishers_ws
with primary at LDS.pubs2
with all tables named 'publishers'
(pub_id char(4),
pub_name varchar(40),
state char(2))
primary key (pub_id)
send standby replication definition columns
replicate minimal columns
```

If a replication definition already exists for the same primary table and is marked for use by the standby, creating a new replication definition using **send standby** (or altering another replication definition) unmarks the previous replication definition as being used by the standby.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Use Replication Definitions and Subscriptions* for more information about using replication definitions with warm standby applications.

See also

- *Specify the Primary Key* on page 320
- *Replicate the Minimal Set of Columns* on page 321

Specify text, untext, and image Column Replication

There are several requirements if you want to create a replication definition for a table that contains `text`, `untext`, or `image` columns datatypes.

You must:

- Include each `text`, `untext`, or `image` column that you want to replicate in the column list, and
- Include each column in the optional clauses **`replicate_if_changed`** or **`always_replicate`**. In each clause, enclose the names of the `text` and `image` columns in parentheses. For multiple columns, separate each column from the next with a comma.
- Ensure that each `text`, `untext`, or `image` column has a corresponding status in Adaptive Server.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > What Information is Replicated for ASE Warm Standby Application? > Replicate text, untext, image, and rawobject Data.*

See also

- *Replicate text, untext, image, and rawobject Columns* on page 336

Specify Computed Column Replication

To create a replication definition for a computed columns use the base column datatype in the replication definition for materialized columns. Do not include virtual columns in the replication definition.

Specify rawobject and rawobject in row Column Replication

You can include Java columns in a replication definition. Replication Server replicates Java columns as either `rawobject` or `rawobject in row` datatypes.

To create a replication definition for a table that contains Java datatypes:

- Include each `rawobject` or `rawobject in row` column that you want to replicate in the column list, and
- Include each `rawobject` column in the optional clauses: **`replicate_if_changed`** or **`always_replicate`**.

In each clause, enclose the names of the `rawobject` columns in parentheses. For multiple columns, separate each column from the next with a comma.

Note: `rawobject in row` columns do not have replication status.

- Ensure that each `rawobject` column has a corresponding status in Adaptive Server.

See also

- *Replicate text, untext, image, and rawobject Columns* on page 336

Specify Column-level Datatype Translations

You can specify column-level datatype translations in the replication definition.

Sybase provides a set of datatype definitions that you install using instructions from the *Replication Server Configuration Guide* for your platform.

- The *declared_datatype* defines the datatype of the value delivered to the Replication Server from the Replication Agent.
- The *published_datatype* defines the datatype of the value after a column-level translation. If there is no column level translation defined using the **map to** clause of the replication definition, then the published datatype defaults to the declared datatype.
- The *delivered_datatype* is the datatype of the value as delivered to the replicate database. This may be different from the published datatype if there are class level translations defined for the function-string class used for the replicate data server's connection. If there is no class level translation defined, then the delivered datatype defaults to the published datatype.

See also

- *Translate Datatypes Using HDS* on page 372

Create Replication Definitions Using Extended Limits

Replication Server version 12.5 and later can replicate wider columns, wider parameters, and larger numbers of columns than earlier versions. It can also handle wider data rows and wider messages.

Replication Server supports the extended limits capabilities of Adaptive Server version 12.5 and later. See the Adaptive Server documentation for more information. For information about using Replication Server extended limits with non-Sybase data servers, see the documentation for your Sybase Replication Agent and the *Replication Server Heterogeneous Guide*.

Before You Use Extended Limits

To use extended limits, ensure that the site, route, and Adaptive Server version are upgraded to the supported version level.

To use extended limits, make sure that both the primary and replicate Replication Server are upgraded to site version 12.5 or later, which automatically sets the LTL version to 400. In addition, make sure that all routes using extended limits are set to 12.5 or later. If you are using Adaptive Server, make sure that both the primary and replicate databases are set to version 12.5 or later. Both the primary and replicate databases must be configured for the same page size.

See the Replication Server white paper “Using Adaptive Server Enterprise version 12.5 with Replication Server version 12.1 and earlier: Schema-length and compatibility issues.”

See also

- *Replication Definition Restrictions in Mixed-Version Systems* on page 329

Use Extended Limits

You can create replication definitions using extended limits for both replicate and standby databases.

Extended limits are defined as:

- Wide columns – data rows containing more than 255 to a maximum of 32768 bytes.
- More columns – replication definitions containing more than 250 columns in a replication definition.
- Wide data – data rows up to the size of the data page on the data server. Adaptive Server version 12.5 and later supports page sizes of 2K, 4K, 8K, and 16K.
- Wide messages – messages larger than 16K.

Wide Columns

Replication Server can replicate wide columns containing `char`, `varchar`, `binary`, `univarchar`, `unichar`, `unitext` or Java `inrow` data to a maximum of 32768 bytes. Maximum column width on each system may vary; it is a function of the total number of columns and the page size of the data server.

You can use wide columns as primary keys and searchable columns and in replication definition **where** clauses.

Note: The maximum number of bytes in the **where** clause of a subscription or article is 255 bytes. You cannot use wide columns in the **where** clause of subscriptions or articles.

More Columns

There is no explicit limit on the number of columns in a replication definition. Replication Server does not limit the number of primary key or searchable columns.

Replication Server uses primary key columns to build **where** clauses of SQL statements for the data server. Consider data server limitations when determining the actual number of columns available for primary keys in replication definitions.

Similarly, although Replication Server imposes no limits on the number of searchable columns in a replication definition, the number of columns in the **where** clause of a subscription or article may also be constrained by data server limitations.

Wide Data

Data rows can equal the size of the data page on the data server. Adaptive Server version 12.5 and later supports page sizes of 2K, 4K, 8K, and 16K.

Wide Messages

Replication Server copies data rows as messages in stable queues managed by the SQM. These messages contain before and after images of replicated data rows as well as other information.

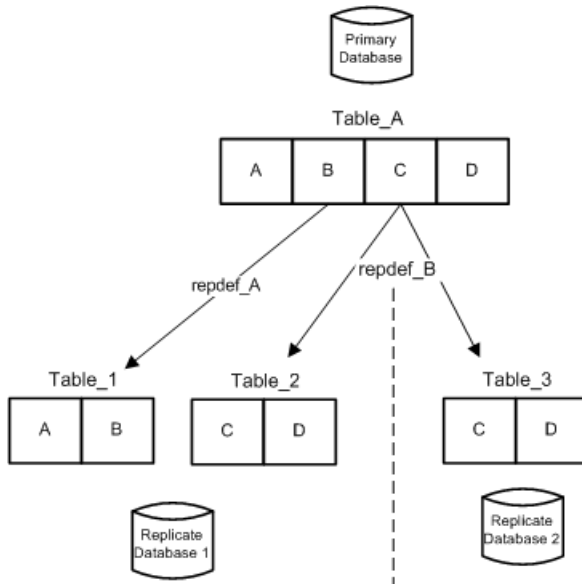
They require significantly more space than the data rows on which they are based. With extended limits, messages can span blocks and are no longer limited to 16K.

Create Multiple Replication Definitions Per Table

You can create multiple replication definitions for the same primary table and customize each one so that it can be subscribed to by a replicate table whose characteristics are different from those of the primary table or from other replicate tables.

For example, you can create two separate replication definitions for the same primary table, one that replicates columns A and B, and another that replicates columns C and D. Each subscribing site receives only the columns that it needs as illustrated in the "Using Multiple Replication Definitions from One Primary Table" figure.

Figure 19: Using Multiple Replication Definitions from One Primary Table



In addition to describing the primary table, each replication definition can specify a smaller number of columns, different column names, different published datatypes, or a different table name for a replicate table. Replicate tables that match the specified characteristics can subscribe to the replication definition.

Different replication definitions created for the same primary table must use the same declared column datatype (unless the datatype is `rawobject` or `rawobject in row`) and the same **null** and **not null** status for `text`, `unitext`, and `image` columns. Use **alter replication definition** to alter a column's datatype or null status.

You can change replication status using **alter replication definition**. For example, you can change the replication status of `text`, `unitext`, and `image` columns from

Manage Replicated Tables

replicate_if_changed to **always_replicate**. The replication status for the column will also change for other replication definitions for the same primary table.

See also

- *Alter Column Datatypes* on page 360

Restrictions to Multiple Replication Definitions

There are restrictions that apply when you have multiple replication definitions for the same primary table.

- A replicate database can subscribe to multiple replication definitions. However, a replicate table can subscribe to only one replication definition of a particular primary table.
- A pre-version 12.0 Replication Server may not subscribe to replication definitions that either declare columns with User-Defined Datatypes or employ column-level translations.
- Different replication definitions created for the same primary table must use the same column datatype (unless the datatype is `rawobject` or `rawobject in row`) and, for `text`, `unitext`, `image`, and `rawobject` columns, the same **null** or **not null** status and the same replication status.
- You cannot create multiple replication definitions for a single primary stored procedure.
- Multiple replication definitions for one primary table are only supported in Replication Server version 11.5 and later; however, one replication definition can be marked and propagated to a Replication Server of a previous version, if compatible; that is, has the same primary and replicate table names, same primary and replicate column names, and does not include table owner name.

See also

- *Specify Column Names and Datatypes* on page 317
- *Replication Definition Restrictions in Mixed-Version Systems* on page 329

Replication Definitions and Function Strings

Function strings map Replication Server functions to data server commands for execution in a database.

For each replication definition, the primary Replication Server creates default function strings for the system functions with replication definition scope (**rs_insert**, **rs_update**, **rs_delete**, **rs_select**, and so on). These function strings are distributed with the replication definition to other qualifying Replication Servers with routes from the primary Replication Server.

Some circumstances may require you to create the function strings for system functions (that is, Replication Server does not create them for you).

See *Replication Server Administration Guide Volume 2 > Customize Database Operations* for details on function strings and function-string classes.

See also

- *Specify Database Operations* on page 48

Replication Definition Restrictions in Mixed-Version Systems

There are several restrictions on replication definitions in mixed-version systems.

- If you create or alter a replication definition that includes an identifier longer than 30 characters, only Replication Server 15.0 or later can subscribe to that replication definition.
- After you upgrade a Replication Server to version 15.5 or later, you can only create, alter, or drop replication definitions if the site version is 1550 or later.
- If you execute **alter replication definition** to drop some columns from the replication definition, and there is a subscription to the replication definition from a replicate site with a site version earlier than 1550, the primary Replication Server rejects the **alter replication definition** command.
- Issuing any **alter replication definition** request with the **with DSI_suspended** parameter does not suspend any replicate DSI with site versions earlier than 1550.

See also

- *Mixed-Version Replication Systems* on page 17

Mark Tables for Replication

After you create a replication definition for the table, use **sp_setreptable** to mark the table for replication.

After a table is marked for replication, RepAgent begins forwarding the table's log records to the Replication Server. If you have marked a table for replication, you do not need to mark it again for another replication definition.

Note: Refer to your Replication Agent documentation for instructions on marking tables for replication in non-Sybase data servers.

See also

- *Subscription Example* on page 428

Use the sp_setreptable System Procedure

To use **sp_setreptable**, you must be the Database Owner or the System Administrator for the data server.

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures* for more information about **sp_setreptable** command.

Enable Replication

To designate a primary Adaptive Server table for replication, use **sp_setreptable**.

Log in to the Adaptive Server managing the database for that table and enter:

```
sp_setreptable table_name, 'true'
```

Marking the table in this way specifies that the table name must be unique.

Note: Do not mark a table for replication unless you also create a replication definition for the table in the Replication Server managing that database. The RepAgent will begin forwarding to the Replication Server data for transactions for the affected table. If a replication definition does not exist, Replication Server may report message 32032 and its error log file may fill up. In addition, Replication Server performance may be significantly reduced. Warm standby applications, which do not require replication definitions, are not subject to this problem.

Check Replication Status

You can use **sp_setreptable** to check the replication status for a table or all tables in the database.

To check replication status for the table, enter:

```
sp_setreptable table_name
```

To check replication status for all tables in the database, enter:

```
sp_setreptable
```

Enable SQL Statement Replication

Use **sp_setrepdefmode** to enable and configure SQL statement replication.

sp_setrepdefmode includes options to:

- Enable or disable SQL statement replication for a specific DML operation
- Configure the threshold that must be reached to activate SQL statement replication

The DML operations that apply to SQL statement replication include:

- **U** – update
- **D** – delete
- **I** – insert select

When the table replication mode is set to any combination of **UDI** the RepAgent sends additional information to enable SQL statement replication for the specified DML operation.

For example, to enable SQL statement replication for **update**, **delete** and **insert select** operations on table **t**, enter:

```
sp_setrepdefmode t, 'UDI', 'on'  
go
```

See *Replication Server Administration Guide Volume 2 > Performance Tuning > SQL Statement Replication*.

Enable Replication with owner_on Status

User tables may have the same name but different owners. Adaptive Server allows you to mark a table for replication and specify that table owner information should be considered when identifying the table.

To mark the table for replication with the “owner on” status, log in to Adaptive Server and enter:

```
sp_setreptable table_name, 'true', owner_on
```

At the Replication Server, the replication definition for the table must identify the table owner. For example, if you set owner status for a table to “owner on” with **sp_setreptable**, you must include an owner name when you create the replication definition or Replication Server will be unable to find the correct table at the replicate database.

The owner of the source table and the owner of the destination table can be different.

Note: If you specify “owner off” status for a table, Replication Server does not send table owner information to the replicate site. However, if you are replicating to a standby database, Replication Server sends “dbo” as the table owner.

Refer to your Replication Agent documentation to see if your non-Sybase data server allows user tables with the same name but different owners.

Modify the Owner Status of a Table

You can use the **sp_setrepdefmode** system procedure to change the owner status of a table previously marked for replication.

To change the status of a table already marked for replication to “owner on,” log in to Adaptive Server and enter:

```
sp_setrepdefmode table_name, owner_on
```

To change the status of a table already marked for replication to “owner off,” log in to Adaptive Server and enter:

```
sp_setrepdefmode table_name, owner_off
```

You must reflect a change in owner status by including owner information in the replication definition. Use **create replication definition** at the Replication Server to create a new replication definition that includes the table owner.

Check the Owner Status of a Table

You can use the **sp_setreptable** system procedure to check the owner status of a table.

Enter:

```
sp_setreptable table_name
```

Disable replication

Use `sp_setreptable` with the `false` parameter to turn off replication for the table.

```
sp_setreptable table_name, 'false'
```

Refer to your Replication Agent documentation for instructions on disabling replication in non-Sybase data servers.

Replicate Java Columns

You can replicate Java columns stored in your primary database to your standby and replicate databases.

Replication Server passes Java objects through the replication system in serialized format without altering the Java objects in any way. See *Java in Adaptive Server Enterprise* for complete information about Java classes in the Adaptive Server database.

Restrictions to Replicating Java Columns

Although you prepare replication definitions and subscriptions for Java columns in the usual manner, be aware of the restrictions that apply.

- Both the primary and replicate databases must be Sybase Adaptive Server version 12.0 or later.
- Replication Server does not replicate stored procedures that have Java objects as parameters. However, the effect of such a stored procedure can be duplicated through normal table replication.
- You cannot use Java columns as part of the primary key.
- You cannot evaluate Java columns in subscription expressions because Java columns are not searchable.

Upgrade Considerations

After you have upgraded the current Replication Server and set its site version to the current release, the Replication Manager route upgrade feature copies the replication definitions with Java columns from upstream Replication Servers to the current Replication Server.

Although Replication Server does not propagate replication definitions with Java columns to pre-12.0 version Replication Servers, you can replicate Java columns to older Replication Servers by manipulating function strings.

See also

- *Using Function Strings to Replicate Java Columns to Older Replication Servers* on page 334

Java Datatypes in Replication Server

Java columns pass through the replication system as one of two 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`. This is the default datatype for Java columns in Replication Server. Replication Server handles `rawobject` data in the same way it handles `image` data.
- 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, for example, `char` data is stored. The base datatype of `rawobject in row` is `varbinary(255)`. Replication Server handles `rawobject in row` data in the same way it handles `varbinary(255)` data.

`rawobject` and `rawobject in row` are compatible only with their base datatypes. They are not compatible with each other; that is, you cannot replicate `rawobject` to `rawobject in row` or vice versa.

The Replication Server reconciliation utility `rs_subcmp` treats Java datatypes as their base datatypes. See *Replication Server Reference Manual > Executable Programs > rs_subcmp*.

See also

- *Replicate text, unitext, image, and rawobject Columns* on page 336

Create Replication Definitions for Java Columns

You can create replication definitions for Java columns using **create replication definition** and the `rawobject` and `rawobject in row` datatypes.

When creating a replication definition:

- `rawobject` values have replication status. You can choose whether they are always replicated or replicated only if changed. They also have null status.
- `rawobject in row` values do not have replication or null status.

`rawobject` and `rawobject in row` values:

- Cannot be part of the primary key.
- Cannot be evaluated in subscription expressions. Java columns are not searchable, and thus they cannot be used in a subscription or article **where** clause.

This example creates a sample replication definition `p1` for a table `t` that contains Java columns.

```
create replication definition p1
  with primary at ds.db
  with all tables name t
  (c1 int,
   c2 rawobject null,
```

Manage Replicated Tables

```
c3 rawobject not null,  
c4 rawobject in row)  
primary key (c1)  
replicate_if_changed (c2)  
always_replicate (c3)
```

Columns `c2` and `c3` are `rawobject` columns; they have replication and null status. Column `c4` is a `rawobject in row` column; it does not have replication or null status. Columns `c2`, `c3`, and `c4` are neither part of the primary key nor are they searchable.

See also

- *Replicate text, unitext, image, and rawobject Columns* on page 336

Function Strings for Java Columns

Replication Server uses the `rs_raw_object_serialization` function string to pass Java columns to the replicate database in serialized format, which allows Replication Server to update Java columns directly.

`rs_raw_object_serialization` is contained in `rs_sqlserver_function_class` and `rs_default_function_class`.

When a replication definition references the `rawobject` datatype, Replication Server creates `rs_get_textptr`, `rs_textptr_init`, `rs_datarow_for_writetext`, and `rs_writetext` function strings for each `rawobject` column just as it does for `image` data.

Using Function Strings to Replicate Java Columns to Older Replication Servers

Replication Server version 12.0 does not propagate replication definitions with Java datatypes to pre-12.0 Replication Servers. However, you can replicate Java columns through older Replication Servers if you use the corresponding base datatype (`image` and `varbinary(255)`) and manipulate the `rs_usedb` and `rs_insert` function strings.

The following example illustrates the method.

1. Create tables containing Java columns in the primary and replicate databases:

```
create table tInfo  
  (c1 integer,  
   c2 Name rawobject in row,  
   c3 Address rawobject null,  
   c4 AccountInfo rawobject not null)
```

`Name`, `Address`, and `AccountInfo` are Java classes; `c2`, `c3`, and `c4` are Java columns.

2. Create a replication definition for table `tInfo`.

If at least one of the Replication Server is pre-12.0, you must create a replication definition using the base datatypes for `rawobject in row` (`varbinary(255)`) and `rawobject` (`image`):


```
create replication definition tInfo1
with primary at DS-1.dbase
with all tables name tInfo
(c1 integer,
c2 varbinary(255),
c3 image null,
c4 image not null,
primary key (c1)
...
```

If the primary and replicate databases are managed by Replication Servers version 12.0 or later, a replication definition could be:

```
create replication definition tInfo
with primary at DS-1.dbase
with all tables named tInfo
(c1 integer,
c2 rawobject in row,
c3 rawobject null,
c4 rawobject not null)
primary key (c1)
...
```

- Alter the `rs_usedb` and `rs_insert` function strings for both the primary and replicate database connections. See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Alter Function Strings*.

- For `rs_usedb`:

```
alter function string rs_usedb
for function_string_class_name
output language
'use ?rs_destination_db!sys_raw? set
raw_object_serialization on'
```

This change tells Adaptive Server to return Java column data as serialized binary values at subscription materialization. It also allows Replication Server to insert and update Java columns with serialized binary values.

- For `rs_insert`:

```
alter function string tInfo1.rs_insert
for function_string_class_name
output language
'insert tInfo(c1, c2, c4)
values (?c1!new?, ?c2!new?, 0xaced000574000130)'
```

This change alters `rs_insert` for `tInfo1` to insert the special binary value `0xaced000574000130` in column `c4`. If you do not alter `rs_insert`, the default value may cause Adaptive Server to return a serialization error.

So, you can create two replication definitions for the same table where the columns between the two replication definitions have different primary (declared) datatypes. If the primary Replication Server is version 12.0 or later, you can create both replication definitions `tInfo` and `tInfo1` for table `tInfo`. In this case, replicate Replication Servers version 12.0 and later can subscribe to `tInfo` and Replication Servers version pre-12.0 can subscribe to `tInfo1`.

Note: You cannot use this method to replicate Java columns to standby databases. The standby connection uses the function-string class **rs_default_function_class**, which cannot be altered.

Replicate text, untext, image, and rawobject Columns

Replication Server lets you replicate columns that use the Adaptive Server datatypes `text`, `untext`, `image` and `rawobject`.

- When you replicate `text`, `untext`, `image`, and `rawobject` columns you must specify a compatible replication status for each `text`, `untext`, `image`, and `rawobject` column in both the replication definition and in Adaptive Server.
- You cannot include `text`, `untext`, `image`, or `rawobject` columns as part of the primary key or as searchable columns.
- A unique set of columns must be identified so that `text`, `untext`, `image`, or `rawobject` columns replication will only affect one row at the target database. This column or set of columns must be included in the primary key if a replication definition is used.
- To replicate `text`, `untext`, `image`, and `rawobject` columns, follow these steps:
 - Use **create replication definition** to create a replication definition for a table that contains `text`, `untext`, `image`, or `rawobject` columns.
See [Replication Server Reference Manual > Replication Server Commands > create replication definition](#).
 - Use **sp_setreptable** to mark the table for replication.
See [Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_setreptable](#).
- **sp_setreptable** sets the replication status of `text`, `untext`, `image`, or `rawobject` columns to **always_replicate**.
- If you do not want to replicate some of the `text`, `untext`, `image`, or `rawobject` columns, use **sp_setrepcol** to change the replications status of those columns.
- Use **create subscription** to make subscriptions for the replication definition and begin replicating the `text`, `untext`, `image`, or `rawobject` data.

Note: When you execute an update at the primary database, you can update a `text`, `untext`, `image`, or `rawobject` column and a non-text, non-image, or non-rawobject column—a `char` column, for example—with a single command. When those updates are copied to the replicate database, however, Replicate Server executes two commands, one for `text`, `untext`, `image`, and `rawobject` updates and one for other datatype updates. If you choose to have DSI ignore certain replication errors, only a portion of the row may be replicated, which creates an inconsistent replicate table.

See the *Replication Server Heterogeneous Replication Guide* and the Replication Server Options documentation for information on non-ASE data servers.

See also

- *Change Column Status for text, untext, image, or rawobject Columns* on page 339
- *create subscription Command* on page 420

Replicating Large Objects to Non-ASE Servers Using Enterprise Connect Direct Access

Replication Server replicates large objects such as `text` and `image` to non-ASE servers by passing a **writetext** command to Enterprise Connect Direct Access (ECDA), where it is converted to an **update** statement.

The **writetext** command includes large-object pointers that an **update** statement uses to search and propagate the replicate database. Most data servers have their own unique implementation of updating large objects. Therefore, large-object replication to these servers can become slow and inefficient, often requiring a full table scan of the replicate database for a single update.

Replication Server provides an option to include primary keys with **writetext** commands sent to ECDA. With the primary keys, ECDA can create **update** statements that can efficiently search and replicate the replicate database.

Replication Server introduces the Data Server Interface (DSI) configuration parameter **dsi_alt_writetext**. You can use the **dsi_alt_writetext** to instruct the Replication Server to include a text pointer or a set of primary keys with the **writetext** command.

Note: You need a version of ECDA 15.0 ESD #2 to use this feature.

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

Guidelines for Creating a text, untext, image, or rawobject Replication Definition

There are several guidelines to follow when you create a table replication definition for `text`, `untext`, `image`, or `rawobject` columns.

- Include each `text`, `untext`, `image`, or `rawobject` column that you want to replicate in the column list.
- Specify the datatype for each `text`, `untext`, `image`, or `rawobject` column.
- Specify whether a **null** is allowed for the column in destination tables. This setting must be consistent with the way the source and destination tables are defined.
- Include each column in the optional clauses **replicate_if_changed** or **always_replicate**.

Specify a Null Value for text, unitext, image, and rawobject columns

To specify whether or not a null value is allowed in the replicate table for each `text`, `unitext`, `image`, or `rawobject` column, use **NULL** or **NOT NULL** after the datatype for the column in the replication definition.

This setting must be consistent with the way the primary and replicate tables are defined. For `text`, `unitext`, `image`, and `rawobject` columns, the default is **NOT NULL**, meaning that the replicate table does not accept null values.

If you are using multiple replication definitions, the null value setting should be the same for all replication definitions on a primary table.

Do not specify **NULL** or **NOT NULL** for columns using datatypes other than `text`, `unitext`, `image`, or `rawobject`. Columns with null values cannot be searchable.

The following example replication definition for the table `au_pix` includes a column `pic` of datatype `image`, for which null values are allowed in replicate tables. The `pic` column is included in the **replicate_if_changed** clause.

```
create replication definition au_pix
with primary at TOKYO_DS.pubs2
(au_id char(11),
pic image null)
primary key (au_id)
replicate_if_changed (pic)
```

Mark Tables with text, unitext, image, or rawobject Columns

Use **sp_setreptable** to set the initial replication status for `text`, `unitext`, `image`, and `rawobject` columns in Adaptive Server when you mark the table for replication.

sp_setreptable sets the replication status of `text`, `unitext`, `image`, or `rawobject` columns to **always_replicate**.

Note: If you do not want to replicate `text`, `unitext`, `image`, and `rawobject` columns, use **sp_setreplicate** to mark the table for replication, which sets the replication status of `text`, `unitext`, `image`, and `rawobject` columns to **do_not_replicate**.

If you use **sp_setreptable** to mark a table for replication and the table includes `text`, `unitext`, `image`, and `rawobject` columns, an internal operation needs to be completed for every `text`, `unitext`, `image`, or `rawobject` columns in every data row of the table. This internal modification is performed in a single transaction, and for large tables, this operation may be time-consuming and involve a significant amount of data.

Before you use **sp_setreptable** on a large table that has `text`, `unitext`, `image`, or `rawobject` columns, be sure that you have enough log space for this operation. You may also want to choose a time that will be least disruptive for client applications or replication system administration.

You can speed up the process of marking a table with `text`, `unitext`, `image`, or `rawobject`, if you use the option **use_index**. When using this option, Adaptive Server creates an internal nonclustered index for each `text`, `unitext`, `image`, or `rawobject`, in the table. For example:

```
sp_setreptable aux_pix, true, null, use_index
```

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_setreptable*.

Note: Refer to your Replication Agent documentation for instructions on marking tables for replication in non-Sybase data servers.

Change Column Status for text, unitext, image, or rawobject Columns

Use **sp_setrepcol** to adjust the replication status for `text`, `unitext`, `image`, or `rawobject` columns.

When you mark a table with `text`, `unitext`, `image`, or `rawobject` columns for replication, **sp_setreptable** sets the replication status of `text`, `unitext`, `image`, or `rawobject` columns to **always_replicate**.

The replication status is the same for all replication definitions of a primary table. If you change the replication status for one replication definition with **alter replication definition**, you change the replication status for all replication definitions on the same primary table.

If you do not want to replicate some of the `text`, `unitext`, `image`, or `rawobject` columns in a marked table (or you marked the table using **sp_setreplicate**, which sets the replication status of `text`, `unitext`, `image`, and `rawobject` columns to **do_not_replicate**), use **sp_setrepcol** to adjust the replication status. You can set the replication status for one or all columns to **always_replicate**, **do_not_replicate**, or **replicate_if_changed**.

To use **sp_setrepcol**, you must be the Database Owner or System Administrator for the data server.

Note: If you have marked the database for replication to a standby database using **sp_reptostandby** and marked database tables for replication to a replicate database using **sp_setreptable**, Replication Server copies `text`, `unitext`, `image`, and `rawobject` columns to standby and replicate databases as **always_replicate**. If you want to copy `text`, `unitext`, `image`, and `rawobject` columns as **replicate_if_changed**, use **sp_setrepcol** to adjust the replication status. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > What Information is Replicated for ASE Warm Standby Application? > Replicate text, unitext, image, and rawobject Data* for more information about replicating in a warm standby application.

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_setrepcol*.

text, untext, image, or rawobject Column Replication Status

There are three status clauses to set the replication status of the `text`, `untext`, `image`, or `rawobject` columns.

Table 26. Text , Untext, Image , or Rawobject Column Replication Status

Status clause	Description
<code>always_replicate</code>	Adaptive Server logs replication information for the <code>text</code> , <code>untext</code> , <code>image</code> , or <code>rawobject</code> column whenever any column in the row changes.
<code>replicate_if_changed</code>	Adaptive Server logs replication information for the <code>text</code> , <code>untext</code> , <code>image</code> , or <code>rawobject</code> column only when the column data changes.
<code>do_not_replicate</code>	Adaptive Server does not log replication information for the <code>text</code> , <code>untext</code> , <code>image</code> , or <code>rawobject</code> column.

Enable Column Replication

Use `sp_setrepcol` to mark a column with an `image` or `rawobject` datatype for replication.

Syntax:

```
sp_setrepcol table, column, status
```

For example, to mark the `pic` column of datatype `image` for replication in the table `au_pix`, enter one of the following:

- `sp_setrepcol au_pix, pic, always_replicate`
- `sp_setrepcol au_pix, pic, replicate_if_changed`
- `sp_setrepcol au_pix, pic, replicate_if_changed, use_index`

Disable Column Replication

Use `sp_setrepcol` with the `do_not_replicate` parameter to turn off replication for an `image` or `rawobject` column.

Syntax:

```
sp_setrepcol table, column, do_not_replicate
```

For example, to disable replication of the `pic` column of datatype `image` for replication in the table `au_pix`, enter:

```
sp_setrepcol au_pix, pic, do_not_replicate
```

Enable or Disable Replication for All Columns

To mark all `text`, `unitext`, `image`, and `rawobject` columns in the table with the same replication status, enter “null” instead of a column name.

For example, to mark all `text`, `unitext`, `image`, and `rawobject` columns with the **replicate_if_changed** status:

```
sp_setrepcol table_name, null, replicate_if_changed
```

Execute **sp_setrepcol** with the table name and a `text`, `unitext`, `image`, and `rawobject` column name to display the replication status of the specified column. For example:

```
sp_setrepcol table_name, column_name
```

Execute **sp_setrepcol** with a table name to display the replication status of all of the `text`, `unitext`, `image`, and `rawobject` columns in the table. For example:

```
sp_setrepcol table_name
```

Alter Replication Status for text, unitext, image, or rawobject Columns

When you replicate `text`, `unitext`, `image`, and `rawobject` columns you must specify a compatible replication status for each column in both the replication definition and in Adaptive Server.

If you change the replication status of `text`, `unitext`, `image`, and `rawobject` columns in one table replication definition, the replication status automatically changes in all other replication definitions for the same table that includes those `text`, `unitext`, `image`, and `rawobject` columns.

If the primary database supports the **rs_send_repserver_cmd** stored procedure, you can change the replication status of a `text`, `unitext`, `image`, or `rawobject` column in the primary database by:

1. Using **sp_setrepcol** to change the replication status of the column. For example:

```
sp_setrepcol authors, au_pix, replicate_if_changed
```

2. Using **rs_send_repserver_cmd** to execute an alter replication definition request to change the replication status. For example:

```
exec rs_send_repserver_cmd 'alter replication definition authors replicate_if_changed au_pix'
```

If the primary database does not support **rs_send_repserver_cmd**, or if there are no data rows for the table in the database log:

1. Use **sp_setrepcol** to change the status of the column in the database to the new replication status.
2. Use **alter replication definition** to change the status of the column to the new replication status.

Subscription Issues for replicate_if_changed Status

If you create subscriptions for replication definitions with `text`, `unintext`, `image`, or `rawobject` columns that have the status **replicate_if_changed**, use either bulk materialization or materializing `text`, `unintext`, `image`, or `rawobject` data.

See also

- *Bulk Materialization* on page 404
- *Materialize text, unintext, image, and rawobject Data* on page 431

Function Strings for Replicating text, unintext, and image Data

You must create `rsdatarow_for_writetext`, `rs_get_textptr`, `rs_textptr_init`, and `rs_writetext` function strings for each `text`, `unintext`, or `image` column, if you replicate columns with `text`, `unintext`, or `image` datatypes to a non-Adaptive Server database.

The function string name must be the `text`, `unintext`, or `image` column name for the replication definition.

Note: You cannot replicate `rawobject` or `rawobject in row` columns to non-Sybase databases unless you replicate these columns as their base datatype. The base datatype of `rawobject` is `image`; the base datatype of `rawobject in row` is `varbinary`.

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

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

Replicate Microsoft SQL Server Large Value Data Types

Replication Server supports the replication of Microsoft SQL Server large value data types `varchar(max)`, `nvarchar(max)`, and `varbinary(max)`. These datatypes can each store up to 2,147,483,647 bytes of data..

Replication Server supports Microsoft SQL Server large value data types as user-defined datatypes (UDDs) in the table-level replication environment. Replication Server also supports database-level replication for new Microsoft SQL Server large value data types. The Microsoft SQL Server large value data types are directly mapped to `text`, `unintext`, and `image` Replication Server datatypes.

Table 27. Mapping Microsoft SQL Server Large Value Data Types to Replication Server Base Datatypes

New Microsoft SQL Server Large Value Data Types	Replication Server Base Datatype
<code>varchar(max)</code>	<code>text</code>
<code>nvarchar(max)</code>	<code>unitext</code>
<code>varbinary(max)</code>	<code>image</code>

Limitations of Microsoft SQL Server Large Value Data Types

There are several limitations for replication of Microsoft SQL Server large value data types.

- You can only replicate large value data type columns to Microsoft SQL Server.
- You cannot define as a primary key a large value data type column in the replication definition.
- You cannot define as searchable a large value data type column in the replication definition.
- You cannot replicate stored procedures that include one of the new large value data types as a parameter.
- You cannot use text pointers to manipulate the data of the new large value data types.

In a mixed-version environment, the primary and replicate Replication Server must have a site version of 15.2 and later and an LTL version of 710 and later.

See the *Replication Server Heterogeneous Replication Guide* for datatype issues with non-ASE data servers.

Partial Update of Microsoft SQL Server Large Value Data Types

Partial-update transaction directly writes a character string at a user-defined position of a table column without issuing a **delete** and **replace** command, as would happen in a full update.

Partial update does not support multibyte character set conversion. Partial update support is restricted to Microsoft SQL Server .

Replicate Computed Columns

Computed columns allow you to create an expression and place the result of the expression in a table column.

A computed column is:

- **Materialized** – when its value is computed for each insert or update. A materialized computed column is stored in the same way as regular columns.

Manage Replicated Tables

- Virtual – when its value is computed only when referenced in a query. A virtual computed column is stored in the table or index page.

A computed column expression can be:

- Deterministic – when its value is the same each time it is evaluated.
- Nondeterministic – when its value may be different each time it is evaluated (for example, a date stamp).

See the *Adaptive Server Enterprise System Administration Guide* for more information about creating and managing computed columns.

Replication Server replicates materialized computed columns in DML statements in the same way it replicates other columns; it does not replicate virtual computed columns.

The replication of computed columns is supported by function strings. With Replication Server version 15.0 and later, the class-level function string **rs_set_dml_on_computed** is applied at the replicate database DSI when a connection is established. It issues **set dml_on_computed "on"** after the **use database** statement. If the replicate Adaptive Server is 12.5.x or earlier, the command is ignored.

When you are creating or altering replication definitions for tables containing:

- Deterministic columns – you can choose whether or not to include those columns in the replication definition. Because deterministic columns always realize the same value, you can create the replication definition without them and allow each replicated insert and update to compute values at the replicate database.
- Nondeterministic columns – you must include nondeterministic computed columns in the replication definition to ensure that the primary and replicate databases remain synchronized.

Replicate Encrypted Columns

With version 15.0, Replication Server supports replication of encrypted columns in Adaptive Server.

Replication Server replicates the encrypted columns from the primary Adaptive Server database, in binary format as ciphertext values, rather than clear text values.

The encryption keys for the primary and the replicate databases must be identical. Use replication to create the encryption key at the replicate database, or use a **dump** and **load** command to ensure that the encryption keys are identical.

Replication Server in a warm standby and in an MSA environment replicates the **create**, **alter**, and **drop** commands of the encryption keys. It also replicates **alter table** to encrypt or decrypt a column. To replicate the **create**, **alter**, and **drop encryption key** DDL commands, the **system_encr_passwd** must be identical for both the primary and the replicate databases.

If the encryption keys are stored in a separate database, ensure that it is synchronized at the same time as the database containing the encrypted columns using those encryption keys.

If data has diverged between the primary and the replicate databases because of earlier encryption keys or because of differences between the initialization vector and the padding, manually sync the data to avoid failures of **update** and **delete** statements.

Restrictions

Replicating encrypted columns has restrictions.

- Text and image columns cannot be encrypted.
- Encrypted columns cannot be used in a **where** clause of a subscription or article because Replication Server receives the value in ciphertext and cannot compare that value to a clear text value. The encrypted columns cannot be searchable columns.
- If an encrypted column is used in a primary key, the encryption key must be defined with INIT_VECTOR NULL and PAD NULL.

The purpose of an initialization vector and padding is to randomize the ciphertext so that two like values encrypted by the same key result in two differing ciphertext strings. If the ciphertext for encrypted data at the primary and the replicate sites differ, then any attempt by the Replication Server to match the before-image from the primary site with the data at the replicate site fails.

If no initialization vector is used, the ciphertext at the source and the target databases exactly match. The matching is required because Replication Server issues a **where** clause on the **update/delete** statements using the ciphertext of the encrypted columns.

- If a table replication definition is not used to replicate the data in a warm standby or MSA environment for a table, all the encrypted columns in that table must be encrypted with keys defined as INIT_VECTOR NULL and PAD NULL.
- All encrypted columns declared in the replication definition must be in `varbinary` format. See *Adaptive Server Enterprise Encrypted Column Users Guide > Encrypting Data > Length of Encrypted Columns* to determine the length of the column.

Note: `rs_subcmp` supports replication of encrypted columns in Adaptive Server.

Replicate Encrypted Data

Replication Server supports the replication of encrypted data.

If your site replicates schema changes, the following DDL statements are replicated:

- **alter encryption key**
- **create table** and **alter table** with extensions for encryption
- **create encryption key**
- **grant** and **revoke create encryption key**
- **grant** and **revoke select** on the key

Manage Replicated Tables

- **grant** and **revoke decrypt** on the column
- **sp_encryption system_encr_passwd**
- **drop encryption key**

The keys are replicated in encrypted form.

If your system does not replicate DDL, manually synchronize encryption keys at the replicate site. The Adaptive Server **ddlgen** utility supports a special form of **create encryption key** for replicating the key's value. See the Adaptive Server documentation for information on the **ddlgen** utility.

For DML replications, the **insert** and **update** commands replicate encrypted columns in encrypted form, which safeguards replicated data while Replication Server processes it in stable queues on disk.

Work with Special Datatypes

Learn how to use the `rs_address`, `identity`, and `timestamp` special datatype columns in replication definitions.

Using the `rs_address` Datatype

The `rs_address` special datatype makes a unique subscription resolution technique possible: 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.

Prerequisites

You must include any columns that use the `rs_address` datatype in the **searchable columns** clause of the replication definition in order to perform bitmap comparison using the **where** clause.

Task

1. To use this subscription resolution method, create tables that use columns of the `int` datatype.
2. Create a replication definition that includes these columns in the column list, but declare the datatype as `rs_address` instead of `int`.

See also

- *Bitmap Subscriptions* on page 433

Replicate identity Columns

`identity` columns store sequential numbers (such as invoice numbers, employee numbers, or record numbers) that are generated automatically. The value of the `identity` column uniquely identifies each row in a table.

`identity` columns use the `numeric` underlying datatype with scale 0, between 1 and 10^{38} -1, inclusive.

`identity` columns are never updated by the **update** command. **update** applied to primary data from a replicate site (using a request function) can never update the `identity` column with `identity` data.

Specify an identity Column in a Replication Definition

To create a replication definition for a table that contains an `identity` column, specify `identity` as the declared datatype for the column or use a column-level translation to specify `identity` as the published datatype for the column.

A replication definition, or multiple replication definitions for the same table, may not publish more than one column that has the datatype `identity`.

If one replication definition publishes a column as `identity`, another replication definition may publish the column as `numeric` and avoid having the extra commands sent with an insert for subscribers to the second replication definition.

How Replication Server Replicates identity Columns

Learn the commands that Replication Server applies when replicating `identity` columns.

Replication Server applies the following command to the replicate table before **insert**:

```
set identity_insert table_name on
```

Replication Server applies the following command to the replicate table after **insert**:

```
set identity_insert table_name off
```

For any table containing 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.

Replicate timestamp Columns

Replication Server supports `timestamp` as a Replication Server datatype.

`timestamp` is defined as `varbinary(8)` with a status bit indicator that differentiates it from `varbinary`. This allows the replication of `timestamp` columns to replicate, standby, and MSA databases.

Manage Replicated Tables

You can also define `timestamp` as a primary key in a replication definition, and a searchable column in a replication definition and a function replication definition.

The `send_timestamp_to_standby` configuration parameter is also added to support `timestamp` replication. When `send_timestamp_to_standby` is enabled and there are no replication definitions, `timestamp` columns are sent to the replicate database.

For any table containing `timestamp` column, the maintenance user must be the owner of the table or must be the “dbo” user of aliased to the “dbo” login name at the replicate database.

Specify a timestamp Column in a Replication Definition

To create a replication definition for a table that contains a `timestamp` column, specify `timestamp` as the declared datatype for the column or use a column-level translation to specify `timestamp` as the published datatype for the column.

A replication definition, or multiple replication definitions for the same table, may not publish more than one column that has the datatype `timestamp`.

Note: The replicate Adaptive Server must be version 15.0.2 or later to support `timestamp` in replication definition.

See *Replication Server Reference Manual > Topics > Datatypes > Date/time, and Date and Time Datatypes*.

Modify Replication Definitions

Learn how to view, alter, and drop replication definitions and learn how Replication Server supports table changes.

In addition, learn how Replication Server supports table changes resulting from the **alter table** command when the table:

- Has subscriptions from a replicate site, or
- Is replicated to the standby database using a replication definition with a **send standby replication definition columns** clause.

Note: Adaptive Server Enterprise version 12.0 allows users to alter existing tables— add non-nullable columns, drop columns, and modify column datatypes.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Use Replication Definitions and Subscriptions > Create Replication Definitions for Warm Standby Databases > alter table Support for Warm Standby* for **alter table** changes that affect warm standby tables with no subscriptions.

See also

- *Modify Replicated Data* on page 362
- *Replication Definition Restrictions in Mixed-Version Systems* on page 329

Maintain Table Schema

To coordinate primary schema or stored procedure changes with the corresponding replication definition changes, use the **rs_send_repserver_cmd** stored procedure to execute replication definition requests at the primary database, while you are changing the primary schema or stored procedure.

Replication Server stores information about table schema in a table replication definition.

When a replication definition is altered, Replication Server may create a new replication definition version. Replication Server processes old data rows that are already in the replication system using the old replication definition version, while processing new data rows entering the Replication Server system using the new replication definition version.

For databases that do not support **rs_send_repserver_cmd**, verify that the data row using the old replication definition is no longer in the database log before you issue the alter replication definition request at the primary Replication Server.

See also

- *Using the Replication Definition Change Request Process* on page 354

View Replication Definitions and Replication Definition Versions

Use the **rs_helprep** and **rs_helpreptable** stored procedures to display information about existing replication definitions, and the **rs_helprepversion** stored procedure to display information on replication definition versions.

See *Replication Server Reference Manual > RSSD Stored Procedures* to use the stored procedures.

Alter Replication Definitions

You may need to alter a replication definition after a column has been added to a primary table or if a destination database requires a column that was not specified in the original replication definition.

In most instances, you alter replication definitions in conjunction with changing database schema in the source or destination table. Be sure to coordinate schema changes between the source and destination sites.

Replication Server may create a new replication definition version when you alter a replication definition. When that happens, Replication Server changes the name of the old replication definition version to a unique name with a “rs_drp” prefix. Replication Server treats replication definitions with names prefixed with “rs_drp” or “rs_in” as internal replication definitions.

Replication Server deletes an internal replication definition when the data associated with the replication definition is no longer in the replication system.

Manage Replicated Tables

To exclude internal replication definitions from a query to the *rs_objects* table, add to the **where** clause of the query:

```
and objname not like 'rs_drp%' and objname not like  
'rs_in%'
```

For example, this query returns the name of all replication definitions created against the *ling.authors* primary table, and excludes the internal replication definitions:

```
select objname from rs_objects where prsid = 16777317  
and dbid = 104 and phys_tablename = 'authors' and  
phys_objowner = 'ling' and objname not like 'rs_drp%'  
and objname not like 'rs_in%
```

See also

- *Modify Replicated Data* on page 362

Replication Definition Change Request Process

When you request changes to replication definitions, Replication Server coordinates the propagation of replication definition changes and data replication automatically.

You can request replication definition changes directly at the primary database using the **alter replication definition**, **alter applied replication definition**, or **alter request function replication definition** commands, while making changes to the database schema.

In Replication Server:

- You can issue a replication definition command directly from a primary database.
- You can use an **alter replication definition** command to instruct Replication Server to suspend the target DSIs after Replication Server applies all data for the old replication definition version at the target database. This provides a window for you to alter the target schema and alter customized function strings before the data for the new replication definition version arrives.
- You can verify that Replication Server can execute a replication definition request successfully by executing the request without changing any data.
- You can instruct Replication Server to skip a failed replication definition request sent by a Replication Agent. When a replication definition command fails at the primary Replication Server, Replication Agent shuts down. If you restart Replication Agent, the failed command executes again unless Replication Server skips the command.

Note: Besides Adaptive Server, Replication Server extends support for **rs_send_repserver_cmd** to supported versions of these non-ASE databases: Microsoft SQL Server and Oracle. Therefore, you cannot use the replication definition change process for IBM DB2. See the *Release Bulletin* for Replication Agent for the supported database versions.

When you issue a replication definition change request, Replication Server determines if there is a need to create a new replication definition version based on the type of change requested. If Replication Server creates a new replication definition version, primary updates before the replication definition change request automatically use the old replication definition version,

while primary updates after the replication definition change request use the new replication definition version.

Note: These procedures in the replication definition change request process do not apply to the **alter function replication definition** command. You must quiesce the Replication Server environment because **alter function replication definition** directly changes the function replication definition.

See also

- *Execute Replication Definition Changes Directly at the Primary Database* on page 351
- *Suspend DSI* on page 352
- *Verify Replication Definition RCL Commands* on page 353
- *Skip Faulty Commands and Error Handling* on page 353
- *Using the Replication Definition Change Request Process* on page 354

Execute Replication Definition Changes Directly at the Primary Database

Use the **rs_send_repserver_cmd** stored procedure to execute replication definition change requests directly at the primary database.

Replication Server supports **rs_send_repserver_cmd** for these replication definition RCL commands:

- **alter replication definition**
- **create replication definition**
- **drop replication definition**
- **alter applied function replication definition**
- **create applied function replication definition**
- **alter request function replication definition**
- **create request function replication definition**

The syntax is:

```
exec rs_send_repserver_cmd 'rs_api'
```

rs_api contains the replication definition you want to change.

For example, to drop the `address`, `city`, `state`, and `zip` columns from the **authors** replication definition at the primary database, enter:

```
exec rs_send_repserver_cmd 'alter replication
definition authors drop address, city, state, zip'
```

When you execute **rs_send_repserver_cmd** at the primary database, the Replication Agent sends the RCL command stored in *rs_api* to the Replication Server, which then executes the RCL command. This ensures that Replication Server replicates the primary data with the proper replication definition version—primary data before the **rs_send_repserver_cmd** is replicated with the old replication definition version, while primary data after the **rs_send_repserver_cmd** is replicated with the new replication definition version.

Manage Replicated Tables

You do not always need to issue replication definition change requests directly from a primary data server. For example, you can execute the alter replication definition request directly from the primary Replication Server in these situations:

- If there is no subscription to the replication definition
- If there are subscriptions to the replication definition, but there is no data in the primary database log for the table or stored procedure
- If you are adding or dropping a searchable column to or from a table replication definition
- If you are adding or dropping a searchable parameter to or from a function replication definition
- If you are altering a replication definition to turn Dynamic SQL on or off

Warning! As Replication Server accepts all commands that Replication Agent sends to Replication Server, you must control access to `rs_send_repserver_cmd` at the primary database.

Changes to rs_locator Table

When you execute `rs_send_repserver_cmd` at the primary database, the Replication Server `rs_locator` system table stores the OQID of the RCL in `type` according to whether the RCL inside `rs_api` is executed:

- Successfully – `type C`
- Unsuccessfully – `type F`

When you execute `sysadmin skip_bad_repserver_cmd` for the RCL that failed, Replication Server updates the `rs_locator` entry to `type S`. Replication Server skips the failed command the next time Replication Agent sends the command.

See `rs_locator` in the *Replication Server Reference Manual*.

Suspend DSI

Use **alter replication definition**, **alter applied function replication definition**, and **alter request function replication definition** with the **with DSI_suspended** option to suspend the standby DSI, if there is one, and each of the subscribing replicate DSI threads.

Replication Server suspends the DSI thread in the standby or replicate database after Replication Server applies all the data for the old replication definition version to the standby or replicate database.

After Replication Server suspends a DSI thread, you can make changes to the target schema or target stored procedures, and to any customized function strings.

When you resume the DSI thread, Replication Server replicates the primary updates using the altered replication definition.

This example shows how to instruct Replication Server to suspend the target DSI after primary data that exists before you execute **alter replication definition** is replicated to the target database:

```
alter replication definition pubs_rep
alter columns with pub_name as pub_name_set
with DSI_suspended
```

You do not need to use **with DSI_suspended** if:

- There is no subscription to the replication definition.
- You do not need to change customized function strings.
- You do not need to change the replicate or standby database schema.

Note: If there is a subscription from a replicate Replication Server with a site version earlier than 1550, the replicate DSI threads for that Replication Server are not suspended.

Verify Replication Definition RCL Commands

Use **admin verify_repserver_cmd** to verify replication definition RCL commands.

When Replication Agent sends a replication definition RCL to Replication Server to execute, and the replication definition RCL fails to execute, Replication Agent shuts down. To avoid this situation, Sybase recommends that you use **admin verify_repserver_cmd** at the primary database to verify that Replication Server can successfully execute a replication definition request before you execute the RCL directly from the primary database. Replication Server returns an error if it cannot successfully execute the request.

Replication Server supports **admin verify_repserver_cmd** for the same replication definition commands as **rs_send_repserver_cmd**:

- **alter replication definition**
- **create replication definition**
- **drop replication definition**
- **alter applied function replication definition**
- **create applied function replication definition**
- **alter request function replication definition**
- **create request function replication definition**

This example shows that **admin verify_repserver_cmd** can detect syntax errors, such as using the “columns” keyword in the command line:

```
admin verify_repserver_cmd, 'alter replication
definition authors drop columns address, city, state, zip
with DSI_suspended'
```

Replication Server returns with a message, such as:

```
Line 1, character 71: Incorrect syntax with the keyword
'columns'.
```

Skip Faulty Commands and Error Handling

Use **sysadmin skip_bad_repserver_cmd** to instruct Replication Server to skip a failed replication definition request the next time Replication Agent starts.

Warning! Use **sysadmin skip_bad_repserver_cmd** carefully. If you execute the command, and then restart the Replication Agent without executing the corrected replication definition

Manage Replicated Tables

command in the primary Replication Server, primary data may replicate using the wrong replication definition version.

In this example, **sysadmin skip_bad_repserver_cmd** instructs Replication Server and Replication Agent to skip the last failed replication definition command in the `pubs2` database of the SYDNEY_DS data server:

```
sysadmin skip_bad_repserver_cmd, SYDNEY_DS, pubs2
```

If the replication definition request you execute at the primary database through **rs_send_repserver_cmd** fails at the primary Replication Server, Replication Agent shuts down. If you restart Replication Agent, the failed command executes again unless Replication Server skips the command. The replication definition request can fail if:

- A subscription is in progress when you execute the request.
- Replication Server cannot find the replication definition or a column.
- There are syntax errors.

If you can resolve the problem that caused Replication Agent to shut down, for example, by waiting for the subscription to complete or correcting the replication definition, you can then restart Replication Agent and Replication Agent reissues the replication definition request.

If the failed replication definition request is caused by an error that cannot be easily fixed, such as a syntax error:

1. Execute the correct replication definition command directly at the primary Replication Server.
2. Execute **sysadmin skip_bad_repserver_cmd** at the primary Replication Server to ensure that Replication Agent and Replication Server skip the most recent failed request that Replication Agent sent.
3. Restart the Replication Agent.

Using the Replication Definition Change Request Process

Use the replication definition change request process that consolidates all the steps for changing a replication definition.

Prerequisites

You can execute the alter replication definition request directly at the primary Replication Server in any of these situations:

- There are no subscriptions for a replication definition.
- There are subscriptions for a replication definition, but there is no data in the primary database log for the primary table or stored procedure.
- You are adding or dropping a searchable column parameter.
- You are turning Dynamic SQL on or off.

Task

1. Group your changes to the primary schema and stored procedure together.
2. Log in to the primary Replication Server.
3. Wait until all subscriptions and articles for the replication definitions are valid, that is, there is no materialization or dematerialization in progress.
4. For each replication definition affected by the primary schema and stored procedure changes, test the replication definition request using **admin verify_repserver_cmd** in the primary Replication Server:

```
admin verify_repserver_cmd, 'alter replication definition authors
drop address, city, state, zip
with DSI_suspended'
```

5. If you have a lot of replication definitions in the replication system, and you have many replication definition change requests, to enhance performance, you can disable **sts_full_cache** for these system tables:

- rs_objects:

```
configure replication server
set sts_full_cache_rs_objects to 'off'
go
```

- rs_columns:

```
configure replication server
set sts_full_cache_rs_columns to 'off'
go
```

- rs_objfunctions:

```
configure replication server
set sts_full_cache_rs_objfunctions to 'off'
go
```

Tip: Execute the Adaptive Server **update statistics** command on the RSSD tables periodically if there are many RSSD changes. For replication definition change requests, such as to create, alter, or drop replication definitions, the affected tables are `rs_objects`, `rs_columns`, and `rs_objfunctions`. For function string change requests, such as to create, alter, or drop function strings, the affected tables are `rs_funcstrings` and `rs_systext`.

6. At the primary database:
 - a) Suspend any data changes to the primary tables and stored procedures that you want to alter.
 - b) Alter the table schema and stored procedures.
 - c) For the replication definition request that you verified in step 4, execute the **rs_send_repserver_cmd** stored procedure. For example:

```
exec rs_send_repserver_cmd 'alter replication definition
authors'
```

Manage Replicated Tables

```
drop address, city, state, zip  
with DSI_suspended'
```

- d) Resume data changes to the primary tables and stored procedures.
7. If you issue any of the replication definition requests using **with DSI_suspended**, then at each subscribing replicate site, wait until Replication Server suspends the DSI, then:
 - a) Alter the replicate table schema or stored procedures, as required.
 - b) Alter any customized function strings as required, and wait for the customized function strings to arrive at the replicate site.
 - c) Resume the replicate DSI.
8. If you changed the **sts_full_cache** setting for any of the tables in step 5, you can change the settings back now.

Sybase strongly recommends that you follow step 4 and use **admin verify_repserver_cmd** to test your replication definition requests. Using **admin verify_repserver_cmd** provides a higher probability that you can avoid a Replication Agent shutdown if a replication definition request command fails when you execute **rs_send_repserver_cmd** at the primary Replication Server.

If any of the schema changes requires you to perform an action at a replicate site, such as changing the replicate schema or a customized function string, then only the very last of the replication definition requests in step 6c affecting the replication definition needs to be issued with **with DSI_suspended**. In some cases, you may need to resume a replicate DSI more than once.

You can skip step 7 if the replicate schema, stored procedures, or customized function strings do not require any changes.

If the Replication Agent shuts down because of a failed replication definition command, you can skip faulty commands to recover Replication Agent.

If the database does not support **rs_send_repserver_cmd**, you need to wait until the primary database log does not have any data rows for the schema that you are changing, and then execute the alter replication definition request at the primary Replication Server.

See also

- *Skip Faulty Commands and Error Handling* on page 353

Changes You Can Make to the Replication Definition

Learn what you can alter in a replication definition.

You can alter the replication definition in one of the following ways:

- Providing a different replicate table name
- Add columns to the columns list
- Provide different replicate column names

- Change the specifications for replicating `text`, `unitext`, `image`, or `rawobject` columns
- Add columns to the primary keys column list
- Remove columns from the primary keys column list
- Add columns to the searchable columns list
- Drop columns from a replication definition
- Drop columns from the searchable columns list
- Change declared or published column datatypes
- Change the specification for replicating minimal columns
- Change how the replication definition is used in replicating to a standby database

Note: Function strings with replication definition scope are not automatically altered when you add columns to a table or to a replication definition. In certain cases, you must alter the function strings manually. See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings*.

Use **alter replication definition** at the primary Replication Server where you created the replication definition, or use **rs_send_repserver_cmd** at the primary database to send the alter replication definition request. The information you can alter in a replication definition is similar to what you provide when you create replication definitions.

- To rename primary or replicate tables, drop and re-create the replication definition.
- To drop or rename primary columns or change column datatypes, drop and re-create all the replication definitions that have the primary columns and follow the replication definition change request process .

See *Replication Server Reference Manual > Replication Server Commands > alter replication definition*.

See also

- *Create Replication Definitions* on page 310
- *Renaming Replicated Tables* on page 363
- *Replication Definition Change Request Process* on page 350

Providing a Different Replicate Table Name

You can alter the replication definition to replicate data from a source table into a destination table with a different name.

For example:

```
alter replication definition publishers
with replicate table named publishers2
```

Change the Specified Columns

Learn how to add or change a column for a replication definition such as, adding or dropping columns and primary keys, altering column datatypes, and specifying a replicate column name that is different from the primary column name.

Adding a Column to a Replication Definition

Learn to add a column to a replication definition.

To add a `char` column named `zip` (for zip code information) to the replication definition **publishers**:

Alter the replication definition to add the `char` column:

```
alter replication definition publishers_rep
add zip char(10)
```

If the column you added to the destination table has a different name from the source column such as `rep_zip_char`, enter:

```
alter replication definition publishers_rep
add zip as rep_zip char(10)
```

See *Replication Server Reference Manual > Replication Server Commands > alter replication definition*.

See also

- *Add and Delete Columns in Source and Destination Tables* on page 363

Drop a Column from a Replication Definition

Learn to drop a column from a replication definition.

In this example, **alter replication definition** drops the `address`, `city`, `state`, and `zip` columns from the **authors** replication definition:

```
alter replication definition authors
drop address, city, state, zip
```

- If there is a subscription from a replicate Replication Server with a site version earlier than 1550, the primary Replication Server rejects the **alter replication definition** request to drop a column.

Note: If you alter a replication definition to drop a column, you may need to reset autocorrection or dynamic SQL settings at replicate Replication Servers with site versions earlier than 1550.

- If there are multiple replication definitions for a primary table, **alter replication definition** drops only the columns from the replication definition you specify in *repdef_name* in the command line.
- The **drop** parameter drops a column or columns from a table replication definition. If a column is part of the primary key or searchable columns, **drop** drops the column from the

primary key list or searchable column list. Replication Server rejects an alter replication definition request to drop a column if the column is:

- The only column
- The only primary key column for the replication definition
- In the **where** clause of a subscription or article
- Before a searchable column which is specified in the **where** clause of an article or subscription.

Dropping a Searchable Column

Learn how to drop a searchable column from a replication definition.

Prerequisites

You can drop searchable columns from a replication definition only if they are not used in subscription or article **where** clause.

Task

1. Use **drop subscription** to remove any subscriptions in which you want the **where** clause to exclude the searchable columns you are dropping.
2. Use **alter replication definition** to drop the searchable column. For example:

```
alter replication definition publishers_rep
drop searchable columns zip
```

This example removes the zip searchable column from the **publishers_rep** replication definition.

See *Replication Server Reference Manual > Replication Server Commands > alter replication definition*.

3. Use **create subscription** to re-create subscriptions to the altered replication definition.

See also

- *drop subscription Command* on page 426
- *create subscription Command* on page 420

Add or Drop Primary Keys

Learn how to add or drop primary keys. Replication Server depends on primary keys to find the correct rows at the replicate or standby table.

To add the column zip as a primary key to the replication definition, enter:

```
alter replication definition publishers_rep
add primary key zip
```

To drop a primary key, enter:

```
alter replication definition publishers_rep
drop primary key zip
```

Manage Replicated Tables

To replace 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.

Warning! If all primary key columns are missing from the primary table, the DSI will shut down.

Alter Column Datatypes

There are several rules and restrictions for altering column datatypes.

- You cannot change the declared column datatype (the datatype in the primary table) if it is used in a **where** clause in a subscription or a **where** clause in an article.
- You cannot change the `rs_address` datatype.
- You can change the column datatype to `text`, `unitext`, `image`, `rawobject`, or `rawobject in row` only if it is not a primary key or searchable column.
- To change the published (replicate) datatype, you must include the declared (primary) datatype of a column (whether it is being changed or not) and the **[map to]** clause.
- Altering a column's datatype and nullability affects the same column across all replication definitions for a table. However, changes between a `rawobject` or `rawobject in row` and its base datatype, affects only the current replication definition.
- Use column nullability changes for `text`, `unitext`, `image`, and `rawobject` columns only.

See also

- *Translate Datatypes Using HDS* on page 372

Provide a Different Replicate Column Name

Learn how to specify a replicate column name that is different from the primary column name.

To replicate data for the source column `zip` into a destination column named `rep_zip`, enter:

```
alter replication definition publishers_rep
alter columns with zip as rep_zip
```

Enter such a command when:

- You alter the existing destination table to add column `rep_zip`.
- You drop and re-create the destination table to contain the column `rep_zip` in place of the original column `zip`.

Change text, unitext, image, and rawobject Replication Status

Use **alter replication definition** to change the replication status of `text`, `unitext`, `image`, and `rawobject` columns in a replication definition.

See also

- *Alter Replication Status for text, unitext, image, or rawobject Columns* on page 341

Adding a Searchable Column

A searchable column lets you create subscriptions based on values in the column.

To add and take advantage of a searchable column:

1. Use **drop subscription** to remove any subscriptions in which you want the **where** clause to include the added searchable column.
2. Use **alter replication definition** to add the searchable column. For example:

```
alter replication definition publishers_rep
add searchable columns zip
```

(This example makes the zip column searchable.)

See *Replication Server Reference Manual > Replication Server Commands > alter replication definition*.

3. Use **create subscription** to re-create subscriptions to the altered replication definition.

See also

- *drop subscription Command* on page 426
- *create subscription Command* on page 420

Change Minimal Column Replication

You can specify that Replication Server uses the minimal number of columns (as opposed to all columns in each row) when it copies update and delete operations.

Enter a command like this:

```
alter replication definition publishers_rep
replicate minimal columns
```

Alter a Replication Definition for Warm Standby Replication

Use **alter replication definition** to specify which replication definition to use to replicate data into a standby database in a warm standby application.

You can also specify whether to replicate all the columns in the table or only the columns in the replication definition.

See *Replication Server Reference Manual > Replication Server Commands > alter replication definition*.

See also

- *Use Replication Definitions with Warm Standby Applications* on page 323

Drop Replication Definitions

Learn how to drop replication definitions.

Before you drop a replication definition, first drop all subscriptions and articles that reference that replication definition.

To access a list of existing subscriptions for a specified replication definition, use **rs_helpsub**. See *Replication Server Reference Manual > RSSD Stored Procedures > rs_helpsub*.

To access a list of existing subscriptions for all replication definitions, use **rs_helpprep**. See *Replication Server Reference Manual > RSSD Stored Procedures > rs_helpprep*.

Enter **drop replication definition** at the primary Replication Server. For example, to drop the **publishers_rep** replication definition, enter a command like this:

```
drop replication definition publishers_rep
```

See *Replication Server Reference Manual > Replication Server Commands > drop replication definition*.

See also

- *drop subscription Command* on page 426
- *Drop Articles* on page 371

Modify Replicated Data

Learn how to modify replicated data and perform related operations to maintain replication.

Before you modify replicated data, carefully review the issues when you planned the replication system. When attempting to modify replicated data, know the dependencies that may exist and the sequence of tasks required to perform the procedure.

See also

- *Manage Subscriptions* on page 397
- *Plan a Replication System* on page 304
- *Using the Replication Definition Change Request Process* on page 354

Add a New Table

To add a new source table, or add a new destination copy for an existing source table, follow the steps in the summary of procedures for replication.

See also

- *Summary of Procedures for Replicating Tables* on page 306

Renaming Replicated Tables

To rename a replicated table, you need to specify the replication definition name and table names and follow the replication definition change request process.

See also

- *Specify the Replication Definition Name and Table Names* on page 316
- *Using the Replication Definition Change Request Process* on page 354

Add and Delete Columns in Source and Destination Tables

Use the replication definition change request process to add or delete columns in the source and destination tables.

See also

- *Using the Replication Definition Change Request Process* on page 354

Change Searchable Columns

You can add searchable columns to a replication definition.

See also

- *Adding a Searchable Column* on page 361

Dropping a Searchable Column

You can drop searchable columns from a replication definition only if the columns are not used in subscription or article **where** clause.

1. Use **drop subscription** to remove any subscriptions in which you want the **where** clause to exclude the searchable columns you are dropping.
2. Use **alter replication definition** to drop the searchable column. For example:

```
alter replication definition publishers_rep
drop searchable columns zip
```

This example removes the `zip` searchable column from the **publishers_rep** replication definition.

See *Replication Server Reference Manual > Replication Server Commands > **alter replication definition***.

3. Use **create subscription** to re-create subscriptions to the altered replication definition.

See also

- *drop subscription Command* on page 426

- *create subscription Command* on page 420

Change Column Datatypes in a Source or Destination Table

Use the replication definition change request process to change column datatypes in a primary and replicate table in a warm standby only setup.

See also

- *Using the Replication Definition Change Request Process* on page 354

Use Publications

A publication lets you collect replication definitions for the same or related tables and stored procedures and then subscribe to them as a group.

You collect replication definitions in a publication at the source Replication Server and subscribe to them with a publication subscription at the destination Replication Server.

With publications, you monitor the status of one publication subscription for a set of tables and procedures.

These steps summarize the procedure for replicating data using publications:

1. Create or select the replication definitions to include in the publication.
2. Create the publication.
3. Create articles that reference the replication definitions you have chosen.
4. Validate the publication.
5. Create a subscription for the publication.

Note: A replicate database can subscribe to different replication definitions of the same primary table directly or through publications—as long as each replication definition references a different table in the replicate database.

To use publications, the primary Replication Server must be version 11.5 or later. To use publication subscriptions, the replicate Replication Server and the route from the primary Replication Server and the replicate Replication Server must be version 11.5 or later.

When you use publications, you create and manage the following objects:

- Articles – replication definition extensions for tables or stored procedures that let you put table or function replication definitions in a publication. Articles may or may not contain **where** clauses, which specify a subset of rows that the replicate database receives.
- Publications – groups of articles from the same primary database.
- Publication subscriptions – subscriptions to a publication. When you create a publication subscription, Replication Server creates a subscription for each of the publication's articles. Publication subscriptions do not contain **where** clauses.

In general, you manage publications and publication subscriptions in the same way as you do replication definitions and subscriptions. However, when you create a publication, you can

specify the subset of rows that the replicate table receives by including a **where** clause in the article—not in the subscription.

You can create and manage publications using the command line.

See also

- *Manage Replicated Functions* on page 379
- *Manage Subscriptions* on page 397

Use Publications to Replicate Data at the Command Line

Learn how to create a publication to replicate data at the command line and prepare the publication for subscription. In addition, learn how to modify and drop a publication and its associated articles and replication definitions.

Commands for Creating and Managing Publications

There are several commands you can use to manage your publications.

All of the commands for managing publications, except **check publication**, are executed at the source Replication Server, where they require **create object** permission. Anyone can execute **check publication** at the source Replication Server—or at the destination Replication Server if the user has the same login and password at both servers.

Table 28. Commands for Managing Publications

Command	Task
create publication	Creates a publication for a group of tables or stored procedures that is to be replicated to one or more subscribing databases.
create article	Creates an article for a publication, allowing you to add one or more where clauses to specify a subset of rows to send to the destination database. The publication and the replication definition on which the article is based must exist before you create an article.
validate publication	Checks that the publication contains at least one article and marks the publication as VALID and ready for subscription.
check publication	Displays the status of the publication and the number of articles it contains.
drop publication	Removes the publication from the <i>rs_publications</i> system table. You can drop the replication definitions associated with the publication if they are <i>not</i> included in other publications or subscriptions.

Command	Task
drop article	Removes the article from the publication and from the <i>rs_articles</i> system table. You can drop the replication definition associated with the article if it is <i>not</i> included in other articles or subscriptions.
rs_helppubs	Displays information about publications and articles.

You can also use RCL commands for working with publication subscriptions.

See also

- *Commands for Creating and Managing Publication Subscriptions* on page 439

Creating Publications and Articles at the Command Line

Learn how to use RCL commands to prepare a publication for subscription and create a subscription against it.

1. At the source Replication Server:

- a) Create or select replication definitions for the tables or stored procedures from which you want to copy data.

The replication definition specifies the source and destination tables or stored procedure and the columns or parameters that are sent to the subscribing database.

- b) Use **create publication** to create the publication that groups the replication definitions.

Publication information is stored in the *rs_publications* system table in the source Replication Server RSSD. It includes the name of the publication, data server, and database. The publication name must be unique for the source Replication Server and database.

The following example creates a publication named **pubs2_pub**. The primary database is *pubs2* managed by the *TOKYO_DS* data server.

```
create publication pubs2_pub
  with primary at TOKYO_DS.pubs2
```

Publication information is not copied to the destination Replication Server until you create a subscription against the publication at the destination Replication Server.

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

- c) Use **create article** to create articles for the publication.

Each article specifies the publication to which it belongs and the table or function replication definition with which it identifies. A publication can contain articles based on the same or different replication definitions. The replication definition and publication must exist when you create the article.

An article includes the names of the publication, the replication definition, and the source data server and database. Article information is stored in the `rs_articles` and `rs_whereclauses` system tables. Each article name must be unique within the publication.

The following example creates an article named **titles_art** based on the replication definition **titles_rep** for the publication **pubs2_pub**.

```
create article titles_art
  for pubs2_pub with primary at TOKYO_DS.pubs2
  with replication definition titles_rep
```

An article can include **where** clauses that specify the rows or parameters to be sent to subscribing databases.

Creating an article invalidates the publication, which makes it ineligible for subscription. After you create an article, you must change the status of the publication to **VALID**, using **validate publication**, before you can create subscriptions against it.

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

- d) Use **validate publication** to change the status of the publication to **VALID**.

When you validate a publication, Replication Server checks that the publication contains at least one article and marks the publication ready for subscription.

Whenever you add or drop an article from a publication, Replication Server invalidates the publication. To mark the publication **VALID**—and ready for subscription—you must execute **validate publication**.

After you validate a publication, you can create a publication subscription against it.

The following example validates the **pubs2_pub** publication. The source database is `pubs2` managed by the `TOKYO_DS` data server.

```
validate publication pubs2_pub
  with primary at TOKYO_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

2. At the destination Replication Server:

- a) Use **create subscription** to create the publication subscription.

When you create a publication subscription, Replication Server creates a subscription for each article in the publication.

See also

- *Create Replication Definitions* on page 310
- *Specifying a where Clause with the create article Command* on page 368
- *Publication Subscriptions* on page 438

Specifying a where Clause with the create article Command

You can include one or more **where** clauses in an article. A **where** clause sets criteria for the column or parameter values that are to be replicated.

If you omit the **where** clause, Replication Server copies all rows for columns specified in the table replication definition or all parameters specified in the function replication definition.

The **where** clause syntax for articles is:

```
[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]...]
...]
```

Each column name in a **where** clause must be listed in the searchable columns list of the table replication definition. The value for each column must have the same datatype as the column to which it is compared.

Note: Each **where** clause in an article is joined by the **or** operator. However, the **!=**, **!<**, **!>**, and **or** operators are not supported inside a **where** clause. The **&** operator is supported only on **rs_address** columns.

The following example creates an article named **titles_art** for the publication named **pubs2_pub**, using a **where** clause that limits replication to rows where the value in the **type** column is 'popular_comp.'

```
create article titles_art
  for pubs2_pub with primary at TOKYO_DS.pubs2
  with replication definition titles_rep
  where type = 'popular_comp'
```

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

See also

- *Bitmap Subscriptions* on page 433
- *Using the rs_address Datatype* on page 346

View Publication Information

Use the **check publication** command and the **rs_helppub** stored procedure to view information about publications and articles.

Display Publication Status and Number of Articles

Use **check publication** to display the number of articles in a publication and its current status.

Any user can execute **check publication** at either the primary or replicate Replication Server. If you execute **check publication** at the replicate Replication Server, you must have the same login and password at the primary and replicate servers.

The following example displays the status and number of articles in the **pubs2_pub** publication.

```
check publication pubs2_pub
  with primary at TOKYO_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands > check publication*.

Display Publication and Article Information

Use the **rs_helppub** stored procedure at either the primary or replicate Replication Server RSSD to display information about a publication and its articles.

Note: Although you can execute **rs_helppub** at the primary or replicate site, **rs_helppub** only displays publication information stored at the site at which it is executed. For example, if you execute **rs_helppub** at the primary site, **rs_helppub** displays information about all publications created at that site. If, however, you execute **rs_helppub** at the replicate site, **rs_helppub** only displays information about publications for which subscriptions have been created at that site.

Here are some examples of using **rs_helppub**:

- To list all publications at a site, enter:

```
rs_helppub
```

The display output includes publication name, status, the primary Replication Server and database names, the number of articles, and the date of the latest change to the publication.

- To display detailed information about a particular publication, enter:

```
rs_helppub publication_name, primary_dataserver,
  primary_db
```

The display output includes the above information and the names of associated articles, replication definitions, and primary and replicate tables. If subscriptions have been created for the publication, the display includes names of the subscriptions, replicate databases, owners, and the date of the latest change to the subscription.

- To display information about a particular article, enter:

```
rs_helppub publication_name, primary_dataserver,
  primary_db, article_name
```

Manage Replicated Tables

The output display includes the name of the publication to which the article belongs, associated replication definitions, status information, and **where** clauses and subscriptions, if any.

See *Replication Server Reference Manual* > *Replication Server Commands* > **rs_helppub**.

Alter Publication Information

If you want to alter an article or publication, you must drop the article or publication and re-create it. You can also make the **where** clauses in an article more selective.

To make the **where** clauses in an article more selective, you can:

- Drop the article and re-create it with altered **where** clauses, or
- Create another article (for the same replication definition), tailoring the **where** clauses to the new row or parameter selection.

See also

- *Drop Publications* on page 370
- *Drop Articles* on page 371

Adding Articles to a Publication

Learn how to add articles to an existing publication.

1. At the source Replication Server:
 - a) Create or select the replication definitions on which the articles are to be based.
 - b) Use **create article** to create new articles.
 - c) Use **validate publication** to validate the publication so that subscriptions can be created for the new articles.
2. At the destination Replication Server:
 - a) To create subscriptions for the new articles, enter **create subscription** or **define subscription** and include the **for new articles** clause.

See also

- *Publication Subscriptions* on page 438

Drop Publications

Use **drop publication** to remove a publication and all of its articles from the system tables.

Before you drop a publication, you must, at the replicate Replication Server, drop all subscriptions created against the publication.

Execute **drop publication** at the Replication Server that manages the source database. You must have **create object** permission.

The following example drops the **pubs2_pub** publication and the articles it contains.

```
drop publication pubs2_pub
    with primary at TOKYO_DS.pubs2
```

Publication information is dropped immediately from the primary Replication Server; it is not dropped from the replicate Replication Server until:

- You attempt to create a subscription against the dropped publication, or
- You enter **check publication** at the replicate Replication Server.

See *Replication Server Reference Manual > Replication Server Commands > drop publication* .

See also

- *Drop Subscriptions for Publications and Articles* on page 445

Drop Associated Replication Definitions

To drop replication definitions associated with the publication, include the **drop_repdef** clause when you execute **drop publication**.

Replication Server drops all replication definitions associated with the publication that are not referenced by other publications or subscriptions.

For example, to drop all replication definitions associated with **pubs2_pub**, enter:

```
drop publication pubs2_pub
    with primary at TOKYO_DS.pubs2
    drop_repdef
```

Drop Articles

Use **drop article** to remove an article from a publication.

Before you drop an article, you must drop subscriptions created against it at the replicate Replication Server.

Execute **drop article** at the Replication Server that manages the source database. You must have **create object** permission.

The following example drops the **titles_art** article for the **pubs2_pub** publication.

```
drop article titles_art
    for pubs2_pub with primary at TOKYO_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands > drop article*.

See also

- *Drop Subscriptions for Publications and Articles* on page 445

Drop the Associated Replication Definition

To drop a replication definition associated with an article, include the **drop_repdef** clause when you execute **drop article**.

Replication Server drops the replication definition if it is not referenced in other publications or subscriptions.

For example, to drop the **pubs2_pub** article and the replication definition it references, enter:

```
drop article titles_art
  for pubs2_pub with primary at TOKYO_DS.pubs2
  drop_repdef
```

Translate Datatypes Using HDS

In a heterogeneous replication system, when information is replicated from one data server to another, values stored at the primary data server must often be altered so that they can be copied successfully to a different datatype at the replicate data server.

User-created function strings can produce these datatype translations, but require significant user input and are limited by the capabilities of the replicate data server.

To make datatype translations more readily available for different data servers, Replication Server provides heterogeneous datatype support (HDS), an easy-to-apply methodology for translating datatypes at the Replication Server. HDS supports selected datatype translations between these data servers:

- Adaptive Server Enterprise
- DB2
- HANA DB
- Oracle
- Microsoft SQL Server
- UDB

When you use HDS, you can choose which columns and datatypes in the primary database are to be translated, and which replicate data servers will receive the translations.

Sources of information:

- See the *Replication Server Heterogeneous Replication Guide*.
- See the *Replication Server Configuration Guide* for your platform for instructions for installing and setting up the objects that enables HDS.
- See the *Replication Server Reference Manual* for descriptions of the function strings.

Overview of Heterogeneous Datatype Support

If you are replicating information between Adaptive Servers, datatype translations are normally unnecessary. However, you can use HDS to perform datatype translations when datatypes differ in the primary and replicate databases.

You can use HDS capabilities when replicating between:

- Adaptive Server databases – one Adaptive Server datatype to another Adaptive Server datatype
- Like non-Sybase databases – for example, DB2 `TIMESTAMP` to DB2 `DATE`
- Heterogeneous non-Sybase databases – Oracle to DB2, for example
- Adaptive Server and non-Sybase databases – Adaptive Server to Oracle, for example

HDS handles incompatibilities between the datatypes of the primary data server and the replicate data server. In general, these incompatibilities are of four types:

- Incompatible ranges – for example, the range of acceptable dates for Sybase `datetime` is January 1, 1753 through December 31, 9999. Another data server, however, may only allow dates from January 1, 0001 through December 31, 9999.
- Incompatible formatting – for example, the primary data server date format is “CCYY-MM-DD,” but the replicate data server requires a date format of “MM/DD/CCYY.”
- Incompatible length – for example, the primary table may have a column with a character length of 10, but the column in the replicate table has a length of 15.
- Incompatible delimiters – for example, Sybase delimits binary data with an “0x” prefix, whereas another data server may surround binary data with single quotation marks.

The Replication Agent for each data server delivers replicate values to Replication Server in a datatype format that Replication Server understands, which includes the literal value, delimiter information, and other datatype attributes. Replication Server handles the value as its base datatype—one of the native Replication Server datatypes described in the *Replication Server Reference Manual*.

You can implement datatype translations in two ways:

- Class-level translations – translate all instances of a datatype for a particular connection. Create a connection specifying a connection profile for your non-ASE database. The connection profile includes the class-level translation. See *Replication Server Reference Manual > Replication Server Commands > create connection using profile*.
- Column-level translations – translate all instances of a column described by a table replication definition.

Use the **map to** clause for column-level translations.

Getting Started with Translating Datatypes

Learn how to set up datatype translations in a heterogeneous replication system.

1. Review the datatype translations available for your primary and replicate data servers. Determine the translations you want and the methods for delivering them:
 - Class-level translations
For lists of supported class-level datatype translations, see the *Heterogeneous Replication Guide*.
 - Column-level translations
2. Set up column-level translations by creating or altering the replication definition.

See also

- *Manage Subscriptions* on page 397
- *Manage Replicated Functions* on page 379

Create Class-Level Translations

Class-level translations for non-ASE data servers are defined in connection profiles and connection profiles install the class-level translations.

See the *Replication Server Heterogeneous Replication Guide* to find a description of the connection profiles for each supported non-ASE database.

System-Defined Variables

Class-level translations change the datatype of system-defined variables as well as column values.

For example, if a class-level translation changes `datetime` to `rs_db2_timestamp`, the `rs_origin_begin_time` system-defined variable, which is `datetime`, is translated to `rs_db2_timestamp` for that connection.

See also

- *Connection Profiles* on page 191

Create Column-Level Translations

Column-level translations affect each replicated instance of a particular column (datatype) and table. They are defined using the **create replication definition** or **alter replication definition** command.

To set up column-level translations, you simply create or alter the replication definition, identifying the column to be translated and its initial and final datatypes using the **map to** option.

- If you are creating a new replication definition, use **create replication definition**.

For lists of supported datatype translations, see the *Heterogeneous Replication Guide*.

- If you are adding or altering a column in an existing table, use **alter replication definition**.

Sybase provides a set of datatype definitions and datatype classes that you can use to modify the datatype of the replicated columns. Each datatype class contains datatype definitions for a particular data server:

- Adaptive Server – **rs_sqlserver_dt_class**
- HANA DB – **rs_hanadb_dt_class**
- IBM DB2 – **rs_db2_dt_class**
- Microsoft SQL Server – **rs_mssql_dt_class**
- Oracle – **rs_oracle_dt_class**
- Sybase IQ – **rs_iq_dt_class**
- UDB – **rs_udb_dt_class**

Datatype classes are not replicated and cannot be modified. Column-level translations are implemented after subscription resolution and before class-level translations.

You can activate a column-level translation for a particular column when you create or alter a table replication definition. The syntax for **create replication definition** with column and datatype variables specified for HDS is:

```
create replication definition replication_definition
with primary at data_server.database
...
(column_name [as replicate_column_name] declared_datatype [null |
not null]
[map to published_datatype])
...
```

where:

- The declared datatype depends on the datatype of the value delivered to the Replication Server from the Replication Agent:
 - If the Replication Agent delivers a native Replication Server datatype, such as `datetime`, to the Replication Server, the declared datatype is the native datatype.
 - Otherwise, the declared datatype must be the datatype definition for the original datatype at the primary database.

For example, the Replication Agent delivers a value in the DB2 `TIMESTAMP` datatype, as a character string with delimiters, to Replication Server. In this case, the declared datatype is the datatype definition `rs_db2_timestamp`.
- The published datatype is the datatype of the column after the column-level translation (and before a class-level translation, if any). The published datatype is normally either a Replication Server native datatype or a datatype definition for the datatype in the replicate database. If the published datatype is omitted from the replication definition, it defaults to the declared datatype.

Both declared and published datatypes have a base datatype. For example, the datatype `rs_db2_timestamp` has a base datatype of `char(26)`; the native datatype `char(26)`

Manage Replicated Tables

also has a base datatype of `char(26)`. A datatype definition describes a non-Sybase datatype in terms of a Replication Server native datatype. The base datatype fixes the maximum and minimum length to be associated with the datatype definition and provides defaults for other datatype attributes. The base datatype defines the delimitation of values for the datatype definition when a value of that type is delivered to Replication Server either in Log Transfer Language (LTL) or in a command executed by a Replication Server administrator such as **create subscription**.

Note: Native datatypes include all datatypes supported by Replication Server. However, you cannot use `text`, `unitext`, `image`, `rawobject`, and `rawobject` in row datatypes for defining a datatype definition; neither can you use these datatypes as the source or target of a translation.

For example, to create a table replication definition `ase_employee_repdef_for_db2` that translates values in the `birthdate` column from `datetime` (`birthdate` primary table datatype) to DB2 `DATE` datatype for the replicate database, log in to the primary Replication Server and enter:

```
create replication definition
  ase_employee_repdef_for_db2
  with primary at ase_server.ase_database
  with all tables named 'employee'
    (empid int,
     first_name char(20),
     last_name char(20),
     ...
     birthdate datetime map to rs_db2_date,
     salary money,
     ...
```

In this example, `birthdate` is the column name, `datetime` is the declared datatype, and `rs_db2_date` is the published datatype. Because the declared datatype is a native datatype, the native and base datatype are the same. That is, the base datatype of `datetime` is `datetime`. The published datatype `rs_db2_date` is a datatype definition for DB2, and its base datatype is `char(10)`.

See also

- *Use Class-Level and Column-Level Translations Together* on page 377

How Datatype Definitions Work

Datatype definitions allow you to translate from one datatype to another without losing valuable information.

When used as the declared datatype, a datatype definition provides the mechanism for capturing both the literal value and its datatype attributes—such as delimiters, range information, precision, scale, length, and maximum and minimum values—and translating them into a native datatype format that Replication Server can process.

When used as a published datatype, a datatype definition takes the value in Replication Server native datatype format, including its attribute information, and translates that information into

a datatype format acceptable to another database, retaining as much information as the published datatype can accommodate.

When data definitions are used for both the declared and published datatypes, both translations take place.

Note: Microsoft SQL Server does not directly support the new unsigned integer types in 15.0 and requires to use a **map to** clause in their replication definitions.

Column-Level Translations and Multiple Replication Definitions

In general, a column declared in multiple replication definitions must use the same declared datatype in each replication definition—although published datatypes can differ.

`rawobject` and `rawobject in row` (Java) columns declared in multiple replication definitions, however, can use either the `rawobject` (or `rawobject in row`) datatype or its base datatype for the declared datatype. For example, you can use `rawobject` and `image` or `rawobject in row` and `varbinary` in multiple replication definitions for the same Java column. See *Java in Adaptive Server Enterprise* for detailed information about Java columns in Adaptive Server.

Use Class-Level and Column-Level Translations Together

If you activate class- and column-level datatype translations for the same column, both are applied.

Column-level translations are performed after subscription resolution and before class-level translations, just prior to delivery to the replicate database.

This order of execution ensures that column-level translations supersede class-level translations. That is, translations for a particular connection (class-level translations) do not affect translations defined for a particular table and column (column-level translations).

Verify Translations

You can verify how translations alter values before you set up column-level or class-level translations.

Use the **admin translate** command to view the results of a particular translation. **admin translate** accepts a value and a source and target datatype and returns the target value. It is most useful with the diagnostic version of Replication Server, which, if the translation fails, allows you to trace the reason for the failure.

The syntax is:

```
admin translate, value, source_datatype, target_datatype
```

where:

- *value* is the literal representation of the value being translated—including delimiters as required by the base datatype of the source datatype.

Manage Replicated Tables

- *source_datatype* is the datatype definition or datatype for the value you want to translate.
- *target_datatype* is the datatype definition or datatype for the value after translation.

If the base datatype of either the source or target datatype requires a length specification, such as `char(26)`, enclose the datatype name in quotes.

For example, to verify the translation of a date from `db2_date` to `datetime`, log in to Replication Server and enter:

```
admin translate, '04/29/1989', db2_date, datetime
```

In this example, *value* is the character string “04/29/1989,” and you must enclose it in single quotes. See *Replication Server Reference Manual > Replication Server Commands > admin translate* for a complete description and further examples.

Manage Replicated Functions

Use replicated functions to replicate the execution of a stored procedure from the source database to the destination database.

When you use function replication, Replication Server replicates the execution of a stored procedure from the source database to the destination database. That is, when a stored procedure is executed at the source database, the Replication Server invokes the execution of another stored procedure at the destination database. The two stored procedures need not have the same name nor perform the same tasks.

See the *Replication Server Design Guide* for information about replication system design issues that concern replicated stored procedures.

See *Replication Server Administration Guide Volume 2 > Asynchronous Procedures* for the distribution of stored procedures associated with table replication definitions. See *Replication Server 15.6 Administration Guide Volume 2 > Pre-15.1 Request Function Replication* for the request function distribution without subscription for versions earlier than 15.1.

You identify the stored procedure at the source and the information that is to be passed to the destination by creating a function replication definition, which specifies:

- The names of the stored procedures at the source and destination databases (if they are different)
- The datatypes and parameters that are to be passed to the destination stored procedure

To satisfy the requirements of distributed applications, Replication Server provides two ways to implement replicate functions. Use:

- An applied function to deliver a transaction to a replicate database by the maintenance user.
- A request function to deliver a transaction to a replicate database by the same user who invokes the stored procedure at the primary database.

The **maint_user** runs the transaction at the replicate database if the function is replicated through **applied function replication definition**. The **origin_user** runs the transaction if the function is replicated through **request function replication definition** at the replicate database.

See also

- *Applied Functions* on page 384
- *Request Functions* on page 387

Prerequisites and Restrictions for Replicated Functions

Before you implement applied or request functions in your replication system, be sure that you have met the prerequisites, and that you understand the restrictions on the use of replicated stored procedures.

Replicated Function Prerequisites

There are several prerequisites to meet when you implement applied or request functions.

- Understand how you will use applied or request functions to meet your application needs. Refer to the *Replication Server Design Guide* for more information.
- Set up a RepAgent at the primary Replication Server by following the procedures involved in managing a replication system and the configuration procedures in the *Replication Server Configuration Guide*.
- Set up routes from the primary Replication Server to the replicate Replication Server.
- Replicated functions can be used with applications that involve fragmented primary data. To do this, create a function replication definition and a stored procedure for each primary fragment. Refer to the *Replication Server Design Guide* for more information about working with fragmented primary data.

In general, the information on replicated functions that is discussed, is based on the Replication Server basic primary copy model, where a single source database distributes data to one or more destination databases.

See also

- *Manage a Replication System* on page 83
- *Manage Routes* on page 165
- *Replication Server Basic Primary Copy Model* on page 8

Replicated Function Restrictions

There are several restrictions when you use replicated stored procedures.

- The names of all replication definitions, including function replication definitions, must be unique in the replication system.
- When you create an applied function replication definition for a primary function in your replication system, make sure that the function does not have an existing function replication definition that satisfies both these conditions:
 - The function replication definition is created using the **create function replication definition** command.
 - The function replication definition is used for the request function replication without subscription in Replication Server 15.0.1 and earlier version.

Otherwise, the existing request function replication will be disabled. See *Replication Server 15.6 Administration Guide Volume 2 > Pre-15.1 Request Function Replication*.

- Replication Server does not support nested transactions—those containing **begin** or **commit** statements—within replicated stored procedures.

If stored procedures with nested stored procedures are marked for replication:

- The RepAgent forwards only the outer stored procedure call to the Replication Server.
- The RepAgent shuts down.
- An error message appears in the Adaptive Server error log.

When the **maint_user** or the replicate database replicates a stored procedure, using **sp_setrepproc** or **sp_setrepligate**, Adaptive Server always executes the stored procedure within a transaction. Even if you have not explicitly executed the replicated stored procedure within a transaction at the primary database, Adaptive Server places an implicit **begin transaction** at the start of the procedure when it is applied by the **maint_user** in the replicate database.

For more information, see **dsi_max_xacts_in_group** in *Replication Server Administration Guide Volume 2 > Performance Tuning > Configuration Parameters that Affect Performance > Connection Parameters that Affect Performance*. If the replicated stored procedure contains such commands as **begin transaction**, **commit transaction**, or **rollback transaction**, errors may result when you execute the procedure. For example, a **rollback transaction** command might roll back to the start of the transaction group, rather than to the nested **begin transaction** command that was the intended rollback point.

- Replicated functions, like Adaptive Server stored procedures, cannot contain parameters with **text** and **image** datatypes. Refer to the *Adaptive Server Enterprise Reference Manual*.
- Adaptive Server logs a replicated stored procedure invocation in the database in which the enclosing transaction was started:
 - If the user does not begin a transaction explicitly, Adaptive Server begins one in the user's current database before the stored procedure is invoked.
 - If the user begins the transaction in one database and then executes a replicated stored procedure in another database, the execution is still logged in the database where the transaction began.
- If a single transaction invokes one or more request functions and executes applied functions or contains data modification language, or a mixed-mode transaction, Replication Server processes the request functions after all the other operations have completed, together in a separate transaction.
- When you use replicated functions and heterogeneous datatype translations:
 - You cannot alter the datatype of a parameter value using **create applied/request function replication definition** or **alter applied/request function replication definition**. However, you can use datatype definitions to declare parameters for applied function replication definitions, which are then subject to class-level translations.

- Replication Server does not perform translations on parameter values for request functions. However, during function-string mapping, the delimiters defined for the parameter values of their declared datatype are used to generate the SQL.

Warning! Do not put a commit statement inside a replicated function as this may cause a duplicate key and make Replication Server recovery fail.

Commands for Managing Function Replication Definitions

Learn the commands you can use to work with function replication definitions.

Table 29. Commands for Managing Function Replication Definitions

Command	Task
drop function replication definition	Removes a function replication definition from the replication system. You must drop all subscriptions for a function replication definition before you can drop the replication definition.
create applied replication definition	Creates an applied function replication definition that describes the stored procedure and its parameters, for both the primary and replicate databases. It also describes the location of the primary data. The maint_user applies the applied function at the replicate site.
create request replication definition	Creates a request function replication definition that describes the stored procedure and its parameters, for both the primary and replicate databases. It also describes the location of the primary data. The same user running the stored procedure at the primary site applies the request function at the replicate site.
alter applied replication definition	<p>Modifies an applied function replication definition, which is created with create applied function replication definition command. For example, it:</p> <ul style="list-style-type: none"> • Specifies a different name for the primary stored procedure invoked at the source database. • Specifies a different name for the stored procedure invoked at the destination database. • Adds parameters or searchable parameters. • Changes how the replication definition is used in replicating to a standby database. <p>If parameters are added, the change applies to all applied function replication definition created for this primary function.</p>

Command	Task
alter request replication definition	<p>Modifies a request function replication definition, which is created with create request function replication definition command. For example, it:</p> <ul style="list-style-type: none"> • Specifies a different name for the primary stored procedure invoked at the source database. • Specifies a different name for the stored procedure invoked at the destination database. • Adds parameters or searchable parameters. • Changes how the replication definition is used in replicating to a standby database. <p>If parameters are added, the change applies to all request function replication definitions created for this primary function.</p>
create function replication definition	<p>Creates a function replication definition that describes the stored procedure, and its parameters, for replication. It also describes the location of the primary data. This command is deprecated and is replaced by create applied function replication definition and create request function replication definition commands.</p>
alter function replication definition	<p>Modifies a function replication definition. For example, it:</p> <ul style="list-style-type: none"> • Specifies a different name for the stored procedure invoked at the destination database • Adds parameters or searchable parameters • Changes how the replication definition is used in replicating to a standby database <p>This command can only be used to modify the function replication definition created with create function replication definition command.</p>
rs_send_repsrvr_cmd	<p>Executes replication definition change requests directly at the primary database.</p>
admin verify_repsrvr_cmd	<p>Verifies that Replication Server can successfully execute a replication definition change request.</p>

See also

- *Modify or Drop Replicated Functions* on page 392
- *Execute Replication Definition Changes Directly at the Primary Database* on page 351
- *Verify Replication Definition RCL Commands* on page 353
- *Commands for Managing Table Replication Definitions* on page 309
- *Subscription Commands* on page 416
- *Implementing an Applied Function* on page 385
- *Implementing a Request Function* on page 388

Use Replicated Functions

A replicated stored procedure is an Adaptive Server stored procedure that you have marked for replication using either **sp_setrepproc** or **sp_setreplicate**. A function replication definition describes the primary and the replicated stored procedure, its parameters, and its location.

You can use these three commands to create a function replication definition:

- **create applied function replication definition**
- **create request function replication definition**
- **create function replication definition** (deprecated)

When you create a function replication definition, Replication Server creates a function, which contains the information in the function replication definition.

When a replicated stored procedure that has its own function replication definition is invoked, its function is transferred from the source to a destination Replication Server. In most cases, the replicated stored procedure is invoked at the primary database and delivered to the replicate database. The only exception is the request function replication with a version earlier than 15.1 without subscription and with such replication definition, where the stored procedure is invoked at the replicate database and delivered to the primary database. In all cases, the primary Replication Server is always the Replication Server where the replication definition is created. This Replication Server controls the primary database. See *Replication Server 15.6 Administration Guide Volume 2 > Pre-15.1 Request Function Replication*.

The function passes parameters to the corresponding stored procedure that is, in turn, invoked in the destination database. A function string translates the function to a syntax that a subscribing database can interpret. When used correctly, function replication can dramatically improve performance because it can encapsulate multiple operations in a single function. Replicated stored procedures do not have to modify any data in order to be replicated.

Applied Functions

Use an applied function to distribute operations performed in a primary database to replicate databases.

Applied functions allow you to realize important performance benefits. For example, if a client application must update a large number of row changes, you can create an applied function that changes many rows, rather than replicating the rows individually.

To use an applied function, you first create a stored procedure in the primary database and a corresponding stored procedure in the replicate database. Use **sp_setrepproc** command to mark the stored procedure to be replicated. At the primary Replication Server, you create an applied function replication definition for the stored procedure. Replicate Replication Servers can subscribe to the function replication definition. When the stored procedure in the primary database is invoked, the replicate Replication Server in turn executes the stored procedure in the subscribing replicate database.

Replication Server does not know in advance what data is needed by the stored procedure at the replicate databases until the execution of the stored procedure is subscribed, thus you must use bulk materialization or the no-materialization method when you subscribe to a function replication definition.

Replication Server executes the stored procedure in the replicate database as the maintenance user, which is consistent with normal data replication.

Implementing an Applied Function

Learn how to create replication objects and execute commands to implement an applied function.

Applied and request functions are very similar. The difference is that the maintenance user executes the applied function at the replicate site, and the same user who executes the stored procedure at the primary database executes the request function at the replicate site.

1. Review the prerequisites and restrictions for applied functions.
2. Set up replicate databases containing replicate tables that the stored procedure will modify.
3. In the primary database, create the stored procedure. The stored procedure may or may not modify primary data. For example, this stored procedure uses the *@pub_name* parameter to update the *pub_name* column of the *publishers* table:

```
create proc update_pubs
@pub_id char(4), @pub_name varchar(40),
as
update publishers
set pub_name = @pub_name
where pub_id = @pub_id
```

4. In the primary database, mark the stored procedure for replicated function delivery, using the **sp_setreproc** system procedure. For example:

```
sp_setreproc update_pubs, 'function'
```

5. In the replicate database, create a stored procedure with the same parameters and datatypes as the stored procedure in the primary database. Typically, the two stored procedures perform the same operations. For example:

```
create proc update_pubs
pub_id char(4), @pub_name varchar(40),
as
update publishers
set pub_name = @pub_name
where pub_id = @pub_id
```

Note: The stored procedure created in the replicate database does not have to have the same name, but must have the same parameter name and datatype.

Warning! A stored procedure invoked in a replicate database in applied function delivery is invoked inside a user-defined transaction. See the *Adaptive Server Enterprise Transact-*

SQL Users Guide for information about operations that are not allowed inside user-defined transactions (for example, the **dump transaction** and **dump database** commands).

Do not mark this stored procedure as replicated. In applied function delivery, only the stored procedure in the primary database is marked as replicated.

However, if the replicate database modifies a standby database, mark the stored procedure in the active and standby replicate databases as replicated if you want to use stored procedure replication to the standby.

6. In the replicate database, grant **execute** permission on the stored procedure to the maintenance user. For example:

```
grant execute on update_pubs to maint_user
```

7. In the primary Replication Server, create an applied function replication definition for the stored procedure. For example:

```
create applied function replication definition
update_pubs_rep
with primary at TOKYO_DS.pubs2
with all functions named update_pubs
(@pub_id char(4), @pub_name varchar(40),
 @state char(2))
searchable parameters (@pub_name, @state)
```

The function replication definition must use the same parameter names and datatypes as the stored procedure in the primary database. You have the option to include only the parameters you want to replicate. If the function replication definition has 0 parameters, you must still include the parentheses for this clause.

If you specify searchable parameters, you can subscribe to function invocations based on the value of the function's parameters. In the preceding example, *@pub_name* and *@state* are searchable parameters. Thus, for example, they can subscribe only to "CA" updates.

If you want to replicate the Adaptive Server `timestamp` datatype, declare the datatype `binary(8)` in the function replication definition.

See *Replication Server Reference Manual > Replication Server Commands > create applied function replication definition*.

8. When you create a function replication definition, Replication Server automatically creates a corresponding function in the default function-string class. See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Work with Functions, Function Strings, and Classes > Functions > User-defined Functions*.

If you are not using a default function-string class or a class inherited from the default or if you want to customize the function's invocation, you need to create a function string for the user-defined function.

9. In the replicate Replication Server, create a subscription to the function replication definition, using **create subscription** and the no-materialization method or **define subscription** and the other bulk materialization commands.

Note: You must use the no-materialization method or bulk materialization—instead of atomic or nonatomic materialization—because Replication Server cannot determine in advance what data is needed for the stored procedure at the replicate site.

For example:

```
create subscription pubs_sub
for update_pubs_rep
with replicate at SYDNEY_DS.pubs2
where @state = 'CA'
without materialization
```

If you specified searchable parameters in the function replication definition, you can subscribe to function invocations based on the value of the function's parameters. In this example, the subscription only receives rows if the value of the *@state* parameter is equal to CA.

See *Replication Server Reference Manual > Replication Server Commands > create subscription*.

10. Verify that all Replication Server and database objects in steps 1 through 9 exist at the appropriate locations. You should now be able to execute the applied function.

See *Replication Server Reference Manual > RSSD Stored Procedures* for information about stored procedures, such as **rs_helpfunc**, that you can use to query the RSSD for information about the replication system.

See also

- *Prerequisites and Restrictions for Replicated Functions* on page 380
- *Mark Stored Procedures for Replication* on page 391
- *Modify or Drop Replicated Functions* on page 392
- *Create or Modify a Function String for a Replicated Function* on page 394
- *Use create subscription command for No Materialization* on page 422

Request Functions

Use a request function to deliver a replicated stored procedure from a primary database to the replicate database through the original user, the same user who invokes the stored procedure at the primary database.

This type of function replication is usually used to enable the remote site to make changes to the central data with the authorized user. For example, a client application at a remote location needs to make changes to the central data. The client application first executes a stored procedure at the remote site—a procedure that may or may not make changes at the remote database. When the stored procedure executes, the replicate Replication Server passes a request function to the central site, where a corresponding stored procedure is invoked that updates the central data. In this example, the remote database is the primary database, while the central database is the replicate database of this request function.

With the primary copy model, a single central database contains all the latest updates. A client application at a remote site can update the central data using request functions. As updates

Manage Replicated Functions

occur at the central table, Replication Server captures the updates and sends them to replicate data servers through applied functions. Execution of stored procedures are stored in the Replication Server stable queues until they can be delivered to the appropriate databases.

To use a request function, create a stored procedure in the remote database and a corresponding stored procedure in the central database. Then, create a request function replication definition at the Replication Server that controls the remote database. The Replication Server that controls the central database can subscribe to this request function replication definition. When the stored procedure in the remote database is invoked, it invokes the stored procedure in the central database.

The Replication Server that manages the central database executes the stored procedure in the central database as the user who executed the stored procedure in the remote database. This guarantees that only authorized users can change central data.

In an application, Replication Server may replicate some or all of the data that is changed in the central database. The changes are distributed to the remote databases managed by Replication Servers that have subscriptions to table replication definitions or as separate applied functions. Either way, the effect of a transaction arrives at the central and then remote databases.

When you use request functions, all updates are made at the central database. This preserves Replication Server primary copy data model and protects the replication system from network failure and excess traffic.

Implementing a Request Function

Learn how to create replication objects and execute commands to implement a request function.

Applied and request functions are very similar. The difference is that the maintenance user executes the applied function at the replicate site, and the same user who executes the stored procedure at the primary database executes the request function at the replicate site.

1. Review the prerequisites and restrictions for request functions.
2. In the replicate Adaptive Server, to manage security, create a login name and password for the user who will execute the stored procedure at the replicate Adaptive Server.
3. In the replicate database, create a replicate stored procedure that updates the real data. For example:

```
create proc update_pubs
@pub_id char(4), @pub_name varchar(40)
as
update publishers
set pub_name = @pub_name
where pub_id = @pub_id
```

Warning! A stored procedure invoked in request function delivery is invoked inside a user-defined transaction. See the *Adaptive Server Enterprise Transact-SQL Users Guide* for information about operations that are not allowed inside user-defined transactions (for example, the **dump transaction** and **dump database** commands).

Do not mark this stored procedure as replicated; however, if this database is also part of a warm standby application, then mark the stored procedure in the active database as replicated if you want to replicate stored procedures to the standby database.

4. In the replicate database, grant **execute** permission on the stored procedure to the same user for whom you created a login name and password in step 2. When the request function is replicated in the replicate database, this user executes it. For example:

```
grant execute on update_pubs to pubs_user
```

5. In the primary database, create a request primary stored procedure with the different name, but the same parameters and datatypes as the stored procedure in the replicate database. The new stored procedure should either do nothing or should display a message to indicate a pending update. Typically, the purpose of this stored procedure is to send a request to other databases, instead of performing any data changes on its own database. For example:

```
create proc update_pubs_request
@pub_id char(4), @pub_name varchar(40)
as
print "Transaction accepted."
```

Note: Use a different name for the stored procedure you create in the replicate and primary databases. In the typical applications, the function will replicate back to the primary database later as an applied function. When you create the request function replication definition in step 8, you must specify the name of the stored procedure in the primary and replicate databases.

6. In the primary database, mark the stored procedure for replicated function delivery using the **sp_setrepproc** system procedure. For example:

```
sp_setrepproc update_pubs_request, 'function'
```

7. In the primary database, grant **execute** permission on the stored procedure to the primary Replication Server user who will invoke it. For example:

```
grant execute on update_pubs_request to pubs_user
```

8. In the primary Replication Server, which manages the request primary stored procedure, create a request function replication definition for this stored procedure. For example:

```
create request function replication definition
  update_pubs_request_rep
with primary at TOKYO_DS.pubs2
with primary function named update_pubs_request
with replicate function named update_pubs
(@pub_id char(4), @pub_name varchar(40)),
 @state char(2))
searchable parameters ( @state)
```

The request function replication definition must use the same parameter names and datatypes as the stored procedure in the replicate database. You have the option to include only the parameters you want to replicate.

See *Replication Server Reference Manual > Replication Server Commands > create request function replication definition*.

9. When you create a function replication definition, Replication Server automatically creates a corresponding user-defined function.

If you are not using a default function string or wish to customize the function's invocation, you need to create a function string for the user-defined function.

10. In the replicate Replication Server, create a subscription to the request function replication definition, using **create subscription** and the no materialization method or **define subscription** and the other bulk materialization commands. For example:

```
create subscription pubs_sub
for update_pubs_request_rep
with replicate at SYDNEY_DS.pubs2
where @state = 'CA'
without materialization
```

If you specified searchable parameters in the function replication definition, you can subscribe to function invocations based on the value of the function's parameters. In this example, the subscription only receives rows if the value of the @state parameter is equal to "CA".

See *Replication Server Reference Manual > Replication Server Commands > create subscription*.

Note: You must use the no-materialization method or bulk materialization, instead of atomic or nonatomic materialization because the Replication Server cannot determine in advance what data is needed for the stored procedure at the replicate site.

11. Verify that all Replication Server and database objects in steps 1 through 10 exist at the appropriate locations. You should now be able to execute the request function at the primary database.

See *Replication Server Reference Manual > RSSD Stored Procedures* for information about stored procedures, such as **rs_helpfunc**, that you can use to query the RSSD for information about the replication system.

See also

- *Prerequisites and Restrictions for Replicated Functions* on page 380
- *Manage Replication Server Security* on page 231
- *Mark Stored Procedures for Replication* on page 391
- *Create or Modify a Function String for a Replicated Function* on page 394
- *Use create subscription command for No Materialization* on page 422

Mark Stored Procedures for Replication

Use the **sp_setrepproc** system procedure to mark stored procedures for replication.

The syntax is:

```
sp_setrepproc [proc_name [, {'false' | 'table' | {'function' [,
{'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. For information on replicating stored procedures associated with table replication definitions, see *Replication Server Administration Guide Volume 2 > Asynchronous Procedures*.

'false' – disables replication for the stored procedure.

'log_current' – logs the execution of the stored procedure you are replicating in the current database, not in the database where the 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 parameter.

Use **sp_setrepproc** according to these guidelines:

- To list all replicated objects in the database, enter **sp_setrepproc** with no parameters.
- To determine the replication status of the stored procedure, enter **sp_setrepproc** with the stored procedure name only.
- Enter **sp_setrepproc** with the stored procedure name and **'function'**, **'table'**, or **'false'** to enable each type of replication or to disable replication for the stored procedure. You must be the System Administrator or the Database Owner to use **sp_setrepproc** to change the replication status of a stored procedure.
- To log the execution of a replicated stored procedure in the database you choose, enter **sp_setrepproc** with **'log_current'**, to log execution in the current database, or **'log_sproc'**, to log execution in the database where the stored procedure resides.

For either applied or request function replication, specify **'function'** to indicate the type of replication definition associated with the stored procedure.

See *Replication Server Reference Manual > Adaptive Server Commands and System Procedures > sp_setrepproc*.

Subscribe to Replicated Functions

You must create subscriptions to function replication definitions for either applied or request functions using **create subscription** and the no-materialization method or **define subscription** and the other commands for bulk materialization: **activate subscription**, **validate subscription**, and **check subscription**.

The only exception is the function replication definitions with versions earlier than 15.1 used for request functions without subscription. See *Replication Server 15.6 Administration Guide Volume 2 > Pre-15.1 Request Function Replication*.

If you specified searchable parameters in the function replication definition, you can subscribe to a function based on the value of its parameters.

You drop subscriptions to function replication definitions using **drop subscription**. They are dropped without purging the replicate data associated with the function. You do not need to specify the **without purge** option.

See *Replication Server Reference Manual > Replication Server Commands*, and related commands, for the full syntax for bulk materialization commands.

See also

- *Bulk Materialization* on page 404

Modify or Drop Replicated Functions

Learn how to modify or drop replicated functions.

Alter a Function Replication Definition

Similar to table replication definitions, when you alter a function replication definition, Replication Server may create a new function replication definition version.

Replication Server processes old data rows that are already in the replication system using the old replication function definition version, while processing new data rows entering the Replication Server system using the new function replication definition version.

See also

- *Replication Definition Change Request Process* on page 350

Modify a Function Replication Definition

Use **alter applied function replication definition** and **alter request function replication definition** to alter the function replication definition to add new parameters, add new searchable parameters, or change the name of the destination stored procedure.

The syntax for the commands:

- **alter applied function replication definition**

```
alter applied function replication definition
function_applied_rep_def
{with replicate function named 'proc_name' |
 add @param_name datatype[, @param_name datatype]... |
 add searchable parameters @param_name[, @param_name]... |
 send standby {all | replication definition} parameters ...|
 }[with DSI_suspended]
```

- **alter request function replication definition**

```
alter request function replication definition
function_request_rep_def
{with replicate function named 'proc_name' |
 add @param_name datatype[, @param_name datatype]... |
 add searchable parameters @param_name[, @param_name]... |
 send standby {all | replication definition} parameters ...|
 }[with DSI_suspended]
```

These two commands are used to change the function replication definitions created by **create applied function replication definition** and **create request function replication definition** command respectively. See *Replication Server Reference Manual > Replication Server Commands* for more information about **alter applied function replication definition** and **alter request function replication definition**.

See *Replication Server 15.6 Administration Guide Volume 2 > Pre-15.1 Request Function Replication* to modify function replication definitions from versions of Replication Server earlier than 15.1.

To add new searchable parameters to the **where** clause of a **define subscription** command, drop and re-create the subscription for the function replication definition for applied and request functions.

See also

- *Create or Modify a Function String for a Replicated Function* on page 394
- *Implementing an Applied Function* on page 385
- *Implementing a Request Function* on page 388

Drop a Function Replication Definition

Use **drop function replication definition** to drop a function replication definition before you recreate it, in order to change or remove parameters, or to rename a function replication definition.

The syntax for this command is:

```
drop function replication definition function_rep_def
```

Re-create the function replication definition after you drop it. When you drop a function replication definition, the associated user-defined function and function string are also dropped. Subscriptions to a function replication definition must be dropped first. You can re-create the subscriptions after you re-create the function replication definition.

Create or Modify a Function String for a Replicated Function

When you create or alter a function replication definition, Replication Server automatically creates or alters the corresponding user-defined function. You must, however, create a function string for the user-defined function if you are not using a class that inherits function strings from **rs_default_function_class**, either directly or indirectly.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Work with Functions, Function Strings, and Classes > Functions > User-defined Functions*.

Create a function string for a user-defined function in the function-string class assigned to the destination database for the replicated function. Use **create function string** at the primary Replication Server to create a function string for a user-defined function.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings > Function Strings and Function-string Classes*.

When you drop a function replication definition, Replication Server always drops the user-defined function and function strings.

You can customize function strings in function-string classes that allow it. In a typical application, the replicated user-defined function passes stored procedure parameter values to the destination Replication Server, and the function string executes the stored procedure with these values in the destination database.

To change the default function string to perform some other action, such as inserting data into an audit log, use the **alter function string** command at the primary Replication Server for the replicated function. The function-string class assigned to the destination database for the replicated function must allow you to customize function strings.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations >* for information on creating and altering function strings. See *Replication Server Reference Manual > Replication Server Commands > create function string*.

Use Publications for Stored Procedures

You can use publications to select stored procedures and/or tables, along with their replication definitions, and subscribe to all of them as a group. Publications let you organize your replication definitions and subscriptions and then monitor their status with a single command.

See also

- *Use Publications* on page 364
- *Publication Subscriptions* on page 438

Manage Subscriptions

Subscriptions resemble SQL **select** statements. They identify the replication definition or publication to which you are subscribing, the source and destination databases and data servers, and the materialization method by which the initial information is to be copied.

You can use a **where** clause to specify a subset of rows or parameters that the destination database receives from the source database.

Materialization is the process of copying data specified by a subscription from a primary database to a replicate database, thereby initializing the replicate table. Replicate data can be transferred over a network, or, for subscriptions involving large amounts of data, loaded initially from media. Initialization from media is called bulk materialization. You use one of four materialization methods, depending on how you want materialization to affect the replication system.

Subscriptions for database replication definitions instruct Replication Server to replicate database objects from the primary to the replicate database. You can choose to replicate or not replicate individual tables, transactions, functions, system stored procedures, and data definition language (DDL). This method, called multisite availability (MSA), requires only a single database replication definition for each primary database and a single subscription for each subscribing database.

Subscriptions for table replication definitions instruct Replication Server to replicate data from primary tables into specified replicate tables. After you have created a replication definition for a primary table, replicate sites must subscribe to the replication definition at the primary database to receive updates.

Subscriptions for function replication definitions require you to use the no-materialization or the bulk materialization methods.

You can subscribe to a group of replication definition articles by subscribing to a publication. Publication subscriptions cannot contain **where** clauses. To subscribe to a subset of rows in an article, you must include a **where** clause when you create the article.

You create subscriptions at the Replication Server managing the database where the replicate data is to be maintained. Your previously created replication definition provides the location of the primary data and defines the structure of the primary table and optionally, of the replicate table, where they differ.

Subscriptions are added to the `rs_subscriptions` system table for both the primary and the replicate Replication Server. The materialization method you select determines how you create subscriptions.

Because a subscription can replicate a large set of rows, materialization can burden the network or impede applications that use the primary or replicate data.

See also

- *Manage Replicated Objects Using Multisite Availability* on page 449
- *Manage Replicated Functions* on page 379

Subscription Materialization Methods

Replication Server offers four methods for creating subscriptions, so you can regulate the effects of materialization on the replication system.

Table 30. Subscription Materialization Methods

Method	Description
Atomic materialization (default)	This method, invoked using the default form of the create subscription command, copies subscription data through the network in a single atomic operation. Replication Server executes the rs_select_with_lock function to retrieve the primary data. This method provides complete consistency throughout the materialization process, but may temporarily obstruct transactions using the primary or replicate data. Do not use this method for large subscriptions if a long-running transaction is unacceptable in the primary database.
Nonatomic materialization	This method, invoked using the create subscription command with the without holdlock clause, is similar to the atomic method, except that consistency constraints during materialization are relaxed to allow clients at the primary database to process transactions during materialization. Replication Server executes the rs_select function to retrieve the primary data. Subscription data is copied in a series of transactions. Because users are allowed to update primary data, this method may result in transactional inconsistency and incomplete data during materialization. When materialization is complete, all inconsistencies are fully corrected. Autocorrection for the replicate table must be enabled to resolve inconsistencies.

Method	Description
Direct load materialization	<p>This method, invoked using the create subscription command with the without holdlock direct_load clause, differs from nonatomic materialization method in that:</p> <ul style="list-style-type: none"> • Data is loaded directly from a primary table into a replicate table. • During the autocorrection phrase, subscriptions fail if there are primary key updates. • Replication of other tables is not suspended during direct load materialization. • The replicate DSI does not go down if there is an error for a subscription. • Multiple tables can be materialized concurrently. • Multiple threads can be used to load data from one primary table to a replicate table.
No materialization	<p>This method, invoked using the create subscription command with the without materialization clause, allows you to create a subscription when the subscription data already exists at the replicate database. You can use this method to create subscriptions to table replication definitions, function replication definitions, and database replication definitions.</p>
Bulk materialization	<p>This method is appropriate when there is too much data to copy through the network. This is a “manual” materialization method that allows you to load the subscription data from media such as magnetic tape.</p> <p>Use this method for subscriptions to database replication definitions and to function replication definitions when data must be initialized at the replicate database.</p> <p>The commands used for bulk materialization are define subscription, activate subscription, and validate subscription.</p>

Atomic Materialization

Atomic materialization is the default materialization method. It is the easiest method to execute and maintains complete data consistency throughout the materialization process.

During atomic materialization, Replication Server logs in to the primary data server as the user creating the subscription and with the password defined at the replicate Replication Server. Therefore, the user must be defined at both the replicate Replication Server and primary database with the same password. The user also needs the same login name and password as the primary Replication Server.

Logged in to the primary data server, the Replication Server selects the subscription rows using a **select with holdlock** operation specified by the **rs_select_with_lock** function. The

holdlock performs a repeatable read, preventing other transactions at the primary site from updating the data until the **select** transaction has completed. The rows are transferred to a materialization queue at the replicate site, where they are applied to the replicate database. You must provide the stable queue with adequate partition space to handle the operation.

Atomic materialization is best for smaller subscriptions where the **select with holdlock** operation does not last long enough to disturb client applications using the primary database. If the subscription selects a large number of rows, you may choose to use nonatomic or bulk materialization, so that clients at the primary database are not affected.

When data already exists at the replicate database, you can use the no-materialization method.

Atomic materialization allows changes to the primary table but effectively delays data server changes until the activation phase of materialization has completed.

Incremental Atomic Materialization

You can avoid long-running transactions at the replicate database by using the **incrementally** option.

The incremental option sends materialization data to the replicate database in a series of transactions, rather than in one large transaction. Otherwise, incremental and non-incremental atomic materialization are identical. Subscription data is available but incomplete until materialization has completed and the subscription is validated.

Rows are removed from the stable queue after they have been successfully inserted, so less partition space is required. You can truncate the database transaction log during materialization, if necessary.

Users at the replicate site will see partial subscription data during materialization, which may invalidate some queries. However, they will have access to inserted rows sooner, which may be beneficial.

The **publishers_rep** replication definition presented in the topics on how to manage replicated tables is used in the following example to create a subscription. The **create subscription** command in the example has no **where** clause, so the subscription causes Replication Server to replicate all the rows in the replication definition. The **incrementally** keyword ensures that the replicate database transaction log does not become full. Clients at the replicate site can be suspended or warned that the **publishers** table is materializing and will contain incomplete data until the process has completed.

```
create subscription publishers_sub
for publishers_rep
with replicate_at SYDNEY_DS.pubs2
incrementally
```

See also

- *Specify the Replication Definition Name and Table Names* on page 316

Nonatomic Materialization

Nonatomic materialization, using the **without holdlock** option of the **create subscription** command, is the same as atomic materialization, except for some differences.

Nonatomic materialization differs from atomic materialization because:

- The data is selected from the primary database without a holdlock. Clients at the primary site can update the data while the **select** operation is in process.
- Transactions are always applied incrementally at the replicate database.

Note: If the **replicate minimal columns** feature is set for the replication definition, you cannot create new subscriptions using nonatomic materialization.

With version 15.2, in nonatomic materialization, Replication Server inserts rows into the replicate database incrementally in 1000-row transactions. Clients at the replicate site that are using the table will see partial subscription data during materialization. This may invalidate some queries. Since the subscription is activated before the data is copied to the replicate database, primary table changes may be applied twice to the replicate table in some circumstances. You must enable autocorrection when you use nonatomic materialization. Autocorrection ensures that a second application of data does not result in an error.

Autocorrection for Nonatomic Materialization

To enable autocorrection, issue the **set autocorrection** command with the **on** option for each replication definition to which you plan to subscribe using nonatomic materialization. .

When using autocorrection, if Replication Server updates or inserts a row in a primary table, it converts the update or insert into a delete followed by an insert, so that the update or insert operation cannot fail because of an existing row.

During nonatomic subscription materialization, Replication Server selects data without a holdlock. After adding the data to the replicate database, Replication Server applies replicated commands. If you enable autocorrection, Replication Server corrects certain temporary inconsistencies that may be caused by selecting the data using the **without holdlock** option.

However, if you execute replicated stored procedures that change subscription data during materialization, autocorrection does not always correct the replicate database. During function calls, autocorrection does not protect against inconsistencies.

After a subscription that uses nonatomic materialization has materialized, you can disable autocorrection for better performance. If you disable autocorrection, you can also specify minimal column replication.

See *Replication Server Reference Manual > Replication Server Commands > set autocorrection*.

See also

- *Replicate the Minimal Set of Columns* on page 321

Direct Load Materialization

Use direct load materialization to materialize data between different kinds of primary and replicate databases.

Direct load materialization differs from other automatic materialization methods:

- This automatic materialization method can be used for Adaptive Server, Microsoft SQL Server, Oracle, and DB2 UDB primary databases.
- Materialization queues are not used with direct load materialization. Data is loaded directly from a primary table into a replicate table.
- Replication is suspended selectively:
 - Replication of tables continues for tables that are not being materialized. DML operations on a primary table that is not being materialized are replicated into the replicate table as the DSI receives them.
 - DML operations on a primary table that is being materialized are stored in a catch-up queue and applied to the replicate table after the initial materialization phase.

Note: When DML operations in a catch-up queue are applied to the replicate table, each **insert** operation is converted into a **delete** followed by an **insert**. Materialization fails when an update changes the primary key.

- Multiple tables can be concurrently materialized with direct load materialization.
- When subscription materialization stops due to an error, regular replication of other tables is not suspended.
- Multiple threads can be used to load data from one primary table to its corresponding replicate table. You can tune this multi-threaded behavior with **max_mat_load_threads**.

Direct load materialization can be used to materialize data:

- from Adaptive Server to HANA DB
- from Microsoft SQL Server to HANA DB
- from Oracle to HANA DB
- from DB2 UDB to HANA DB

Note: Direct load materialization is not supported for materializing data into an Adaptive Server database.

Direct load materialization is enabled through the **direct_load** option of the **create subscription** command. When using direct load materialization, note these restrictions for **create subscription**:

- When the **direct_load** option is used, no other subscription can be created or defined at the same time for the same replicate table.

- The **direct_load** option is for subscriptions to table replication definitions only and is used with **without holdlock**. It cannot be used with the **without materialization** or **incrementally** options.
- The **user** and **password** options are used only with **direct_load**.
- You cannot use the **direct_load** option against a logical or alternate connection. The primary connection in the replication definition and the replicate connection in the subscription must be physical connections.
- The maintenance user of the primary database cannot be used in the **user** and **password** options to create subscriptions.
- You cannot use other automatic materialization methods if the primary database is not Adaptive Server. The only automatic materialization option for Oracle or other databases is direct load materialization.
- You cannot drop a subscription with the **with purge** option if the replicate database is not Adaptive Server.
- The **direct_load** option is available only if the replicate Replication Server site version and route version are 1571100 or later.
- You can use row filtering, name mapping, customized function strings and datatype mapping with subscriptions created using the **direct_load** option.
- Replication Server rejects any attempt to create a subscription with the **direct_load** option if the number of subscriptions being created has reached or exceeded **num_concurrent_subs**.

Primary Database Considerations

- In directly materializing data from a primary database, Replication Server connects to Replication Agent and not directly to the primary database.
- You must have Replication Agent version 15.7.1 SP100 or later to materialize data from a non-Adaptive Server primary database using direct load materialization.
- When invoking the **create subscription** command, Replication Server connects to Replication Agent using the Replication Agent administrator login name.

No Materialization

You can use **create subscription** with the **without materialization** clause to activate a subscription when materialization has already occurred..

To use this method:

- The subscription data must already exist at the replicate database
- The primary and replicate tables must be synchronized
- Activity must be stopped on the primary table so that there are no further updates in the Replication Server stable queues

When creating a subscription with the **without materialization** clause, Replication Server logs in to the primary Replication Server as the user creating the subscription. The user who

Manage Subscriptions

executes **create subscription** must have the same login and password at the primary and replicate Replication Servers.

You can also use **create subscription** with the **without materialization** clause to subscribe to function replication definitions.

Bulk Materialization

With bulk materialization, you manually transfer subscription data between databases.

Use bulk materialization when a subscription is too large to copy through the network. Bulk materialization has very little effect on primary database clients or on the network.

You can use bulk materialization to create subscriptions for function replication definitions for replicated functions.

Bulk materialization uses these commands, which are executed at different points in the materialization process: **define subscription**, **activate subscription**, **validate subscription**.

Use the **check subscription** command to check the status of the subscription.

When you use bulk materialization, you must coordinate:

- The dump to media of the subscription data at the primary site.
- The load from media into the table at the replicate site.
- The application of updates made at the primary site after you make the media dump.

Note: Bulk materialization may require special handling if the primary and replicate databases differ in, for example, table or column names.

Three bulk-materialization methods are available to ensure data consistency between the primary and replicate sites. The method you use depends mainly on whether applications using the primary data can tolerate interruptions.

You can use any of these methods for subscriptions to either table or function replication definitions. With subscriptions to function replication definitions, it may not be obvious which replicate tables will be affected by stored replicated procedures executing in the replicate database.

Before you initiate bulk materialization, you must consider these issues in relation to the existing data in the replicate database.

See also

- *Manage Replicated Functions* on page 379

Summary of Bulk Materialization Methods

There are three bulk materialization method.

Table 31. Summary of Bulk Materialization Methods

Method	Summary of process
Stop updates to the primary table and take a snapshot of the data	Stop all applications from updating the primary data and then retrieve the subscription data from the primary database with a select statement or database dump. Define the subscription and activate it with an option that leaves the DSI suspended for the replicate database. Clients can resume updates to the primary data. After you load the subscription data into the replicate database, you can resume the DSI and validate the subscription.
Simulate atomic materialization	Allow client applications to continue executing transactions against the primary data while the subscription data is retrieved. After defining the subscription, you lock the primary data, retrieve the subscription data, and activate the subscription. The activate subscription command leaves the DSI for the replicate database suspended. After you load the subscription data into the replicate database, you can resume the DSI and validate the subscription.
Simulate non-atomic materialization	This method is the same as simulating atomic materialization, except that you activate the subscription first, and then retrieve the data from the primary database without locking the data. Because of this, the data at the replicate database may be inconsistent with the data at the primary database until the subscription is validated and you are required to enable autocorrection for the replicate data.

Stop Updates at the Primary Database and Take a Snapshot

You can use either of two bulk materialization methods to stop updates at the primary database and take a snapshot.

The two methods are:

- Using the Adaptive Server **mount** command
- Using the Adaptive Server **dump** and **load, select, or bcp** command

Use these methods to retrieve data from the primary database if you are able to suspend updates to the primary data. To maintain consistency, all updates to the primary database are suspended for the duration of the materialization.

Retrieving Data from the Primary Database Using the Adaptive Server mount Command

Use the **mount** command to retrieve data from the primary database.

Prerequisites

You can use this method only if you are using Adaptive Server version 12.5.1 or later, and your primary and replicate databases are identical.

Task

1. Verify that the entire replication system is working.
See [Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server](#).
2. Suspend updates to the data in the primary database by stopping client applications that generate transactions against the primary data directly or indirectly through Replication Servers.
3. Quiesce the replication system components involved with replicating data from the primary Replication Server to the replicate Replication Server.

Use **admin quiesce_for_rsi** at the primary and replicate Replication Servers and any intermediate Replication Servers.
4. Execute the Adaptive Server command **quiesce database tag_name hold db_name list [for external dump] to manifest_file [with override]** to generate the manifest file.
See the [Adaptive Server Enterprise Reference Manual](#).
5. Take a snapshot of the subscription data from the primary database by creating a data dump of both the database and log devices.

You can create a data dump using utilities such as **tar** or **zip**, or the UNIX **dd** command.
6. Use **mount database** to begin loading the snapshot data into the replicate database.
7. Resolve the mismatch of user information between the master database and the loaded user database.
8. Use **rs_init** to add the replicate database to the replication system if it is not already there.
9. Execute **define subscription** at the replicate Replication Server.
10. Use **check subscription** at the primary and at the replicate Replication Servers to verify that the subscription has been defined. When the subscription status is DEFINED at both servers, continue to step 11.
11. Execute **activate subscription** at the replicate Replication Server.
12. Use **check subscription** at the primary and at the replicate Replication Server to verify that the subscription has been activated. When the subscription status is ACTIVE at both servers, continue to step 13.
13. Execute **quiesce release** to resume updates to the primary data.
14. Execute **validate subscription** at the replicate Replication Server.
15. Use **check subscription** at the primary and at the replicate Replication Server to verify that the subscription is VALID at both servers.

When you have completed this procedure, the subscription is created, the replicate data is consistent with the primary data, and replication is active.

Retrieving Data from the Primary Database Using the Adaptive Server dump and load, select, or bcp Commands

Use the Adaptive Server **dump** and **load, select, or bcp** commands and utilities to retrieve data from the primary database.

1. Verify that the entire replication system is working.
See *Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server*.
2. Suspend updates to the data in the primary database by stopping client applications that generate transactions against the primary data.
3. Quiesce the replication system components involved with replicating data from the primary Replication Server to the replicate Replication Server.
Use **admin quiesce_force_rsi** at the primary and replicate Replication Servers and at any intermediate Replication Servers.
4. Execute **suspend log transfer** for the primary database.
5. Take a snapshot of the subscription data from the primary database using a **select** statement or a database dump.
6. Execute **define subscription** at the replicate Replication Server.
7. Use **check subscription** at the primary and at the replicate Replication Servers to verify that the subscription has been defined. When the subscription status is DEFINED at both servers, continue to step 9.
8. Execute the **activate subscription** command, using the **with suspension** clause, at the replicate Replication Server.
9. Use **check subscription** at the primary and at the replicate Replication Server to verify that the subscription has been activated. When the subscription becomes active at the replicate Replication Server, the DSI connection to the replicate Replication Server is suspended.
When the subscription status is ACTIVE at both servers, continue to step 11.
10. Execute **resume log transfer** from the primary database at the primary Replication Server.
11. Begin loading the snapshot data into the replicate database.

Note: While you wait for the data to finish loading in the replicate database, you can continue with the next step.

12. Execute **validate subscription** at the replicate Replication Server to validate the subscription.
13. Use **check subscription** at the primary and at the replicate Replication Server to verify that the subscription status is VALID for both servers.
14. When the snapshot data has finished loading in the replicate database, execute **resume connection** to resume the connection to the replicate database.

Manage Subscriptions

When you have completed this procedure, the subscription is created, the replicate data is consistent with the primary data, and replication is active.

Simulate Atomic Materialization

Use simulate atomic materialization as a bulk materialization method when you cannot suspend updates to the primary database.

This method ensures replicated data consistency by retrieving the subscription data, activating the subscription, and suspending the DSI connection to the replicate database all in one transaction at the primary data server.

1. Verify that the entire replication system is working.
See Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server.
2. Execute the **define subscription** command at the replicate Replication Server.
3. Wait for the subscription to be defined at both the primary and replicate Replication Servers. Execute the **check subscription** command at both the primary and replicate Replication Servers to verify that the subscription status is DEFINED.
4. Activate the subscription by executing a single transaction as shown in this sample transaction that includes **select with holdlock** and the **rs_marker** stored procedure.

```
begin transaction
select from table with holdlock
where search_conditions
execute rs_marker
'activate subscription subid
with suspension'
commit transaction
```

subid is an integer that identifies the subscription. The *subid* for a subscription can be found in the *subid* field of the *rs_subscriptions* system table in the RSSD. After the subscription is defined, you can find its *subid* by executing the following query in the RSSD of the primary or replicate Replication Server:

```
select subid from rs_subscriptions
where subname = 'subscription'
and dbid in (select connid from rs_databases
where dbname = 'rep_connection_dbname'
and dsname = 'rep_connection_dsname')
```

where *rep_connection_dbname* and *rep_connection_dsname* can be for the default or alternate connections.

5. Wait for the subscription to become active at both the primary and replicate Replication Servers. Execute the **check subscription** command at the replicate Replication Server to verify that the subscription status is ACTIVE. When the subscription status is ACTIVE at the replicate Replication Server, the database connection for the replicate database has been suspended.

6. As soon as the subscription becomes active at the primary Replication Server, retrieve the data from the primary database using a **select** or a database dump.
7. Find the ID number (*subid*) for the subscription by querying the `rs_subscriptions` system table.

```
select subid from rs_subscriptions
where subname = 'subscription'
and dbid in (select connid from rs_databases
where dbname = 'rep_connection_dbname'
and dsname = 'rep_connection_dsname')
```

8. Execute the **rs_marker** stored procedure in the primary database:

```
rs_marker 'validate subscription subid'
```

Warning! Be sure that you execute the **rs_marker** stored procedure with the correct *subid* number for the subscription. The *subid* column in the `rs_subscriptions` system table contains the unique ID number for each subscription. Entering any other number or character string may cause serious problems.

For more information on **rs_marker** see *Replication Server Reference Manual*.

9. Load the subscription data into the replicate database.
10. Enable autocorrection for the replication definition at the replicate database.
11. Use the **resume connection** command to resume the database connection for the replicate database.
12. Wait for the subscription to become valid at both the primary and replicate Replication Servers. Execute the **check subscription** command at the replicate Replication Server to verify that the subscription status is **VALID**. Once the subscription status is **VALID**, the replicate data is consistent with the primary data.
13. Disable autocorrection for the replicate database.

Now the subscription is created and replication is active.

See also

- *Autocorrection for Nonatomic Materialization* on page 401
- *Replicating Tables in the Example Replication System* on page 428

Simulate Nonatomic Materialization

Use simulate nonatomic materialization as a bulk materialization method when you cannot suspend updates to the primary database or if you cannot lock the primary data during the **select** or **dump** operation that retrieves the subscription data.

This method allows a period of flux at the replicate site during which the replicate data may be inconsistent with the primary data. By the time the subscription becomes **VALID**, however, the data should be consistent. You must set autocorrection on during materialization so that inconsistencies resulting from continuing updates in the primary database can be resolved without errors.

Warning! Do not use this method if the **replicate minimal columns** feature is set for the replication definition or if you execute applied functions or applied stored procedures from the primary database to modify data in the replicate database. In both cases, autocorrection cannot resolve the inconsistencies.

1. Verify that the entire replication system is working.
See [Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server](#).
2. Execute the **define subscription** command at the replicate Replication Server.
3. Wait for the subscription to be defined at both the primary and replicate Replication Servers. Execute the **check subscription** command at both the primary and replicate Replication Servers to verify that the subscription status is DEFINED.
4. Execute the **activate subscription** command, using the **with suspension** clause, at the replicate Replication Server.
5. Wait for the subscription to become active at both the primary and replicate Replication Servers. Execute the **check subscription** command at the replicate Replication Server to verify that the subscription status is ACTIVE. When the subscription status is ACTIVE at the replicate Replication Server, the database connection for the replicate database has been suspended.
6. As soon as the subscription becomes active at the primary Replication Server, retrieve the data from the primary database using a **select** or a database dump.
7. Find the ID number (*subid*) for the subscription by querying the `rs_subscriptions` system table.

```
select subid from rs_subscriptions
where subname = 'subscription'
and dbid in (select connid from rs_databases
where dbname = 'rep_connection_dbname'
and dsname = 'rep_connection_dsname')
```

where `rep_connection_dbname` and `rep_connection_dsname` can be for the default or alternate connections.

8. Execute the **rs_marker** stored procedure in the primary database:

```
rs_marker 'validate subscription subid'
```

Warning! Be sure that you execute the **rs_marker** stored procedure with the correct *subid* number for the subscription. The `subid` column in the `rs_subscriptions` system table contains the unique ID number for each subscription. Entering any other number or character string may cause serious problems.

For more information on **rs_marker** see *Replication Server Reference Manual*.

9. Load the subscription data into the replicate database.
10. Enable autocorrection for the replication definition at the replicate database.

11. Use the **resume connection** command to resume the database connection for the replicate database.
12. Wait for the subscription to become valid at both the primary and replicate Replication Servers. Execute the **check subscription** command at the replicate Replication Server to verify that the subscription status is VALID. Once the subscription status is VALID, the replicate data is consistent with the primary data.
13. Disable autocorrection for the replicate database.

Now the subscription is created and replication is active.

See also

- *Autocorrection for Nonatomic Materialization* on page 401
- *Replicating Tables in the Example Replication System* on page 428

Bulk Copy-in and Subscription Materialization

Use bulk copy-in to improve the performance of subscription materialization.

When **dsi_bulk_copy** is on, Replication Server uses bulk copy-in to materialize the subscriptions if the number of **insert** commands in each transaction exceeds **dsi_bulk_threshold**.

Note: In normal replication, bulk operation is disabled for a table if **autocorrection** is on. However, in materialization, bulk operation is applied even when **autocorrection** is enabled, if **dsi_bulk_threshold** is reached and the materialization is not a nonatomic subscription recovering from failure.

See also

- *Configuration Parameters Affecting Physical Database Connections* on page 201

Dematerialization Processing

Dematerialization removes subscriptions and, optionally, data from the replicate database. Dematerialization also removes subscription information from the RSSDs at the primary and replicate sites.

Dropping a subscription causes Replication Server to stop sending changes from a primary database to a replicate database. You can use the **drop subscription** command to drop subscriptions for either table or function replication definitions.

drop subscription removes the subscription from the RSSDs of the primary and replicate Replication Servers.

When you drop a subscription to a table replication definition, you can specify that Replication Server delete the subscription's rows from the replicate database. Or, you can delete the rows manually.

Manage Subscriptions

When you drop a subscription to a function replication definition, the replicate data associated with the function is not deleted from the replicate database.

There are two methods of dematerialization:

- **with purge** dematerialization, which selectively deletes rows not used by other subscriptions
- **without purge** dematerialization, which allows you to manually delete rows in replicate tables

In either case, the primary Replication Server stops sending data for the dropped subscription, if the data is not included in other subscriptions at the same replicate site.

Note: For user defined datatypes: Subscriptions that specify columns subject to class- or column-level translations in the **where** clause cannot be dematerialized automatically. You must use the bulk or no-materialization method.

Dematerialization and Purging Rows

Use the **with purge** clause when you want to delete rows replicated by the subscriptions you are dropping.

Use the **incrementally** option to delete rows in 1000-row increments. The maintenance user for the replicate database must have **select** permission on the table to use this option.

Dematerializing a subscription and purging rows from the replicate table uses function strings for the **rs_select** or **rs_select_with_lock** system functions. You may be required to create a function string for these system functions.

- If the connection for the replicate database uses a function-string class with default-generated function strings or a function-string class inherited from such a class, Replication Server generates a corresponding default function string for the **rs_select_with_lock** or **rs_select** functions.
- If the connection uses any other function-string class, you must create the function string, with an input template that matches the subscription's **where** clause. Use the **create function string** command.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Function-string Classes* for details.

If you are using a function-string class in which you can customize function strings, you can replace an existing default or custom function string with one that performs a select operation that your application requires, using the **alter function string** command.

For more information on creating or altering **rs_select** and **rs_select_with_lock** function strings, see *Replication Server Administration Guide Volume 2 > Customize Database Operations > Manage Function Strings*.

Dematerialization Without Purging Rows

Dropping a subscription using the **without purge** option leaves the rows replicated by the subscription in the replicate table.

Subscriptions to function replication definitions are dropped automatically using the **without purge** option. You do not need to specify this option. You must, however, specify this option if you want to keep the rows in the replicate table. If you want to manually delete rows, you must use the **with suspension** option as well.

Monitor Materialization and Dematerialization

Subscriptions pass through phases before they are fully set up or removed from the replication system. Learn the phases for setting up a subscription or removing subscription data.

The phases for setting up a subscription are:

- Definition – **create subscription** or **define subscription** add the subscription to the RSSD for the primary and replicate Replication Servers.
- Activation – takes place after subscription resolution. The primary Replication Server adds the subscription to the Subscription Resolution Engine (SRE). The SRE compares log records to the current subscriptions to determine where changes to replicated tables must be distributed.
- Materialization – for atomic and nonatomic subscriptions, the primary Replication Server retrieves subscription data from the primary database and copies it to the replicate Replication Server to be applied to the replicate database.
- Validation – both the primary and replicate Replication Server completely materialize the subscription and verify it is consistent with the primary data.

The phases for removing subscription data, using the **drop subscription** command, are:

- Dematerialization – stops sending updates for the subscription to the replicate database and, if the **with purge** clause is specified, deletes the subscription data from the replicate database (if the data is not included in other subscriptions). If the **without purge** clause is specified, then Replication Server does not delete the data from the replicate database.
- Removal – deletes the subscription from the RSSD for both the primary and replicate Replication Servers.

Materialization or dematerialization can fail during any of these stages. This is why you need to monitor the progress of a subscription using the **check subscription** command.

In addition to the **check subscription** command, you can use the **admin who** command to check the status of the Replication Server threads processing the subscription. For atomic and nonatomic materialization, Replication Server builds a materialization queue that contains rows to be added to the replicate table. The **admin who, sqm** command can monitor queue activity, and the **admin who, dsi** command can show you whether the DSI thread is running.

See *Replication Server Reference Manual > Replication Server Commands* for information about executing **admin who** and interpreting its results.

See the *Replication Server Troubleshooting Guide* for comprehensive troubleshooting information that details the status of a subscription and suggested actions.

See also

- *check subscription Command* on page 425

Verifying that the Replication System Is Ready Before You Create Subscriptions

Before creating subscriptions, verify that the replication system is ready.

1. Verify that all components in the replication system are working. See *Replication Server Administration Guide Volume 2 > Verify and Monitor Replication Server > Verify a Replication System* for details.
2. Make sure the following database objects and permissions exist:
 - One or more replication definitions exist for the primary table.
 - The primary table is marked as replicated with **sp_setreptable** or **sp_reptostandby** for warm standby applications.
 - A table corresponding to the replication definition exists in the replicate database. Its columns must match those specified for the replicate database in the replication definition. Its datatypes must match the corresponding primary columns.
This table must also be visible to the user creating the subscription and the user maintaining it. If an owner name is included in the replication definition, the table must be visible to all database users. If an owner name is not included in the replication definition, the easiest way to make the table accessible is to have the Database Owner create it.
 - The replicate database maintenance user must have:
select, **insert**, **update**, and **delete** permissions on the replicate table, and **execute** permission for functions used in replication.
If the subscription for the table includes the **subscribe to truncate table** clause, the maintenance user must have **replication_role**, **sa_role**, or alias the Database Owner.
3. Make sure that you meet recommended guidelines for the character sets and sort orders used throughout your replication system. These play an important role in processing subscriptions, and they must be consistent everywhere for subscriptions to be valid. Refer to the *Replication Server Design Guide* for guidelines.
4. Choose one of the subscription materialization methods and verify the following requirements for your chosen method:

- For nonatomic materialization, you must enable autocorrection for the replicate table. See *Replication Server Reference Manual > Replication Server Commands* for **set autocorrection** command details.

If the **replicate minimal columns** feature is set for the replication definition, you cannot create new subscriptions using nonatomic materialization.

- For atomic and nonatomic materialization:

A default function-string class or a function-string class inherited from a default function-string class generates default function strings for the **rs_select_with_lock** or **rs_select** functions. If you use other function-string classes, you must create function strings for the **rs_select_with_lock** or **rs_select** functions, with an input template that matches the subscription's **where** clause.

To modify **rs_select** or **rs_select_with_lock**, use a function from the function string class associated with the primary database connection, not the functions in the replicate database connection.

See *Replication Server Administration Guide Volume 2 > Customize Database Operations > Function-string Classes* and *Replication Server Administration Guide Volume 2 > Customize Database Operations > Managing Function Strings > Using Input Templates* for details.

- If you are creating a subscription with either atomic or non-atomic materialization methods and you have columns that require quoted identifiers in the replication definition, you must configure the primary connection to use quoted identifiers.
 - If you are using quoted identifiers with a custom function string that includes a quoted constant, **create subscription** without a quoted constant or **without materialization** clause. Otherwise, during subscription materialization the quoted constant causes a query failure. The replicate data server identifies the quoted constant as a column instead of a constant.
5. When you create subscriptions, use the login name of a regular user. Do not create subscriptions as the maintenance user.

Make sure the user creating the subscription has the following login names and permissions:

- Same login name and password at the replicate Replication Server, the primary Replication Server, and the primary data server. If you are using bulk materialization or the no-materialization method, you are not required to have a login name for the primary data server.
- **select** permission on the primary table. This does not apply if you are using bulk materialization or no materialization.
- **execute** permission on the **rs_marker** stored procedure in the primary database or no materialization.
- **create object** or **sa** permission in the replicate Replication Server.
- **primary subscribe**, **create object**, or **sa** permission in the primary Replication Server.

See also

- *Autocorrection for Nonatomic Materialization* on page 401
- *Subscription Materialization Methods* on page 398

Subscription Commands

Learn how to manage subscriptions with RCL commands.

You can use RCL commands or Sybase Central to:

- Create subscriptions for atomic and nonatomic materialization and for the no-materialization method.
- Define, activate, and validate subscriptions for bulk materialization.
- Check the status of subscriptions during the materialization process.
- Drop subscriptions to initiate the dematerialization process.
- Enable replication of the **truncate table** command when you create or define a subscription.

You can use a **where** clause to control which table rows or function invocations to replicate. The **where** clause can specify only the searchable columns or searchable parameters specified in the table or function replication definition. If you do not provide a **where** clause, all the rows of the replication definition's columns, or all the function invocations, are replicated.

If you are using Adaptive Server Enterprise version 11.5 or later, you can include the **subscribe to truncate table** keywords to reproduce execution of the **truncate table** command at the destination database.

Table 32. Commands for Managing Subscriptions

Command	Task
create subscription	<p>Creates a subscription that transfers the initial version of the replicated data using either:</p> <ul style="list-style-type: none"> • Atomic materialization, which copies the initial version of the data for a subscription as a single transaction, or • Nonatomic materialization, which copies the data in a series of transactions. Users at the replicate site can see some of the data before it all arrives. Replication Server does not create a materialization queue for the entire set of subscription data. • Direct load materialization, which copies the primary data directly to the replicate site. Users at the replicate site can see some of the data before it all arrives. Replication Server does not create a materialization queue, but it creates a catchup queue to hold DML for the primary table before the subscription status is VALID. The content of the queue is applied to the replicate Replication Server after the data initially selected from the primary is applied to the replicate database. <p>Use create subscription with the without materialization clause to activate a subscription for which the initial version of the replicated data already exists at the replicate database. You can also use create subscription to create subscriptions for table replication definitions. Use create subscription, with the without materialization clause, for function replication definitions.</p>
define subscription	<p>The first step in bulk materialization defines a subscription. You can use define subscription and the other bulk materialization commands to create subscriptions for either table or function replication definitions when you create replicated functions. You must transfer data manually, as necessary. Data replication begins after materialization is complete and a subscription is activated and validated. Use check subscription to verify subscription status.</p>
activate subscription	<p>Second step in bulk materialization. Activates a subscription at both primary and replicate Replication Servers. This causes the primary Replication Server to start sending changes to the subscription's data to the replicate Replication Server.</p>
validate subscription	<p>Third step in bulk materialization. Changes the subscription status at both the primary and replicate sites to VALID.</p>
check subscription	<p>Verifies the status of a subscription at both the primary and replicate sites. Use this command with all types of subscription materialization.</p>
drop subscription	<p>Removes a subscription from the replication system. For subscriptions to table replication definitions, optionally removes subscription rows from the replicate table in a process known as dematerialization.</p>

See also

- *Manage Replicated Functions* on page 379
- *Enable Replication of truncate table* on page 419

- *Commands for Managing Table Replication Definitions* on page 309
- *Commands for Managing Function Replication Definitions* on page 382

Use the where Clause

You can include one **where** clause in a subscription to create selective subscriptions.

The **where** clause syntax is a subset of the Transact-SQL **where** clause. The **where** clause is supported by the **create subscription** and **define subscription** commands for subscriptions to replication definitions. The supported syntax is the same for both commands and allows you to create very selective subscriptions. It is designed for efficient processing by the Subscription Resolution Engine in Replication Server.

Note: You cannot evaluate a Java column in a subscription expression. Thus, you cannot include a column of type `rawobject` or `rawobject in row` in a subscription **where** clause.

For subscriptions to table replication definitions, the **where** clause syntax is:

```
where column_name{< | > | <= | >= | = | &} value
  [and column_name{< | > | <= | >= | = | &}
    value]...
```

For subscriptions to function replication definitions, the **where** clause syntax is:

```
where @param_name
  {< | > | <= | >= | = | &} value
  [and @param_name
    {< | > | <= | >= | = | &} value]...
```

See *Replication Server Reference Manual > Topics > Datatypes* for entry formats for values for different datatypes.

Note: The **!=**, **!<**, **!>**, and **or** operators are not supported. You can create multiple subscriptions instead of using the **or** operator. The **&** operator is supported only on `rs_address` columns.

Each column name in a **where** clause must be listed in the **searchable columns** list of the table or function replication definition. The *value* for each column must have the same datatype as the column to which it is compared.

For example, to specify that you want to subscribe to data where `state = CA` for table replication definition **publishers_rep**, you would enter:

```
create subscription publishers_sub1
for publishers_rep
with replicate at SYDNEY_DS.pubs2
where state = 'CA'
```

Note: The maximum size of a **where** clause in a **create subscription** statement is 255 characters.

To subscribe to data in `publishers`, where `state = CA` or `state = MA`, you would need to create two subscriptions. In addition to the preceding command, you would enter:

```
create subscription publishers_sub2
for publishers_rep
with replicate at SYDNEY_DS.pubs2
where state = 'MA'
```

Note: When you use a **where** clause with a subscription for heterogeneous datatype columns subject to class- or column-level translations, you must make sure that you use the correct datatype in the comparison.

See also

- *Bitmap Subscriptions* on page 433
- *Subscriptions for Columns with Heterogeneous Datatypes* on page 432
- *Using the rs_address Datatype* on page 346

Enable Replication of truncate table

You can enable replication of the **truncate table** command to particular destination database tables when you create or define a subscription, if you are using Adaptive Server Enterprise version 11.5 or later.

The **truncate table** command can truncate one or more partitions. Replication Server will recreate the same command executed at the primary database. This requires the replicate site to have the same partition names, otherwise, DSI shuts down.

You have an option to skip **truncate table** and apply appropriate action at the replicate site, or use **rs_truncate** function string to customize the action in the replicate site. Replication Agent sends this command once the LTL version is set to 700.

To create or define a subscription that enables replication of **truncate table**, log in to Replication Server and enter:

```
create subscription subscription
  for table_rep_def
  with replicate at data_server.database
  ...
  subscribe to truncate table
```

When **truncate table** executes at the destination database, Adaptive Server deallocates whole data pages. It does not delete rows one at a time.

Note: 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 the maintenance user **replication_role**, you cannot 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.

Warm standby applications can copy the execution of **truncate table** to standby databases without a subscription. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Alter Warm Standby Database Connections > Alter Logical Connections > Replicate truncate table to Standby Databases*.

Manage Subscriptions

See *Replication Server Reference Manual > Replication Server Commands* for complete command syntax and usage guidelines for **define subscription** and **create subscription**.

Change the Status of subscribe to truncate table

Use **sysadmin apply_truncate_table** to change the status of “subscribe to truncate table” for all subscriptions on a replicate table.

All subscriptions for a replicate table in a particular database must either support or not support replication of **truncate table**. You cannot create a subscription that enables replication of **truncate table** if all existing subscriptions for that table do not support replication of **truncate table**.

For example, to turn on replication of **truncate table** for all subscriptions to a replicate table, log in to the replicate Replication Server and execute this command at the **isql** prompt:

```
sysadmin apply_truncate_table, data_server,  
database, {table_owner|'|'|""}, table_name'on'
```

where *data_server* is the name of the replicate data server, *database* is the name of the replicate database managed by the data server, *table_owner* is the owner of the replicate table, and *table_name* is the name of the replicate table.

If you specified a replicate table owner in the replication definition, you must also specify a table owner with the **sysadmin apply_truncate_table** command. 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.

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

create subscription Command

Use the **create subscription** command to replicate data by subscribing to a replication definition.

There are three methods for creating a subscription:

- Atomic
- Nonatomic
- Direct load materialization
- No materialization

You can use a **where** clause to replicate only certain rows from the primary table, based on values for the searchable columns specified in the table replication definition. If you do not provide a **where** clause, all rows are replicated.

If you are using Adaptive Server Enterprise version 11.5 or later, you can include the **subscribe to truncate table** keywords to reproduce execution of the **truncate table** command at the destination database.

See *Replication Server Reference Manual* > *Replication Server Commands* > **create subscription**.

See also

- *Use the where Clause* on page 418
- *Enable Replication of truncate table* on page 419
- *Create Replication Definitions* on page 310

Use create subscription command for Atomic Materialization

Use **create subscription** to enable atomic materialization.

To create a subscription with atomic materialization, execute the **create subscription** command at the Replication Server managing the database where the data is to be replicated. The syntax is:

```
create subscription subscription
  for table_rep_def
  with replicate at data_server.database
  [where search_conditions]
  [incrementally]
  [subscribe to truncate table]
```

where *subscription* is the name of the subscription to activate, *table_rep_def* is the name of the table replication definition you are subscribing to, and *data_server.database* identifies the replicate database.

The *subscription* name must be unique for the replication definition and replicate database.

Subscribing to function replication definitions requires you to use **define subscription** (the bulk materialization method) or **create subscription** with the **without materialization** clause (the no materialization method).

If you use the optional keyword **incrementally**, Replication Server initializes the subscription by sending 1000-row batches of inserts.

If you do not use the keyword **incrementally**, Replication Server inserts all of the subscription rows at the replicate database in a single transaction. All of the rows are held in a stable queue at the replicate Replication Server at one time, and there must be enough partition space to accommodate them. Also, the transaction log for the replicate database must have enough space to log the transaction.

Use create subscription command for Nonatomic Materialization

Use **create subscription** with the **without holdlock** clause to create a subscription with nonatomic materialization.

The syntax is:

```
create subscription subscription
  for table_rep_def
```

Manage Subscriptions

```
with replicate at data_server.database
[where search_conditions]
without holdlock
[subscribe to truncate table]
```

where *subscription* is the name of the subscription to activate, *table_rep_def* is the name of the table replication definition you are subscribing to, and *data_server.database* identifies the replicate database.

Nonatomic materialization is always incremental.

Monitor materialization and dematerialization as clients at the replicate site should be suspended or warned that the data in the replicate table is incomplete and possibly inconsistent until all the subscription data has materialized.

See also

- *Monitor Materialization and Dematerialization* on page 413

Use create subscription command for Direct Load Materialization

Use **create subscription** with the **direct_load** option to create a subscription with direct load materialization.

The syntax is:

```
create subscription subscription
  for table_rep_def
  with replicate at data_server.database
  [where search_conditions]
  without holdlock
  direct_load
```

where *subscription* is the name of the subscription to activate, *table_rep_def* is the name of the table replication definition you are subscribing to, and *data_server.database* identifies the replicate database.

Unlike other automatic materialization methods, data selected from the primary table is loaded directly to the replicate database—no materialization queue is used—as soon as it is returned from the **select** command. This option is for subscriptions to table replication definitions only and is used with **without holdlock**. It cannot be used with **without materialization** or **incrementally**.

Use create subscription command for No Materialization

Execute **create subscription** with the **without materialization** clause at the Replication Server managing the replicate database to create a subscription that does not initialize the subscription data.

The syntax for **create subscription** for no materialization is:

```
create subscription subscription
for {table_rep_def | function_rep_def | publication pub |
  database replication definition db_repdef
  with primary at server_name.db }
```



```
with replicate at server_name.db
[where search_conditions]
without materialization
[subscribe to truncate table]
```

where *subscription* is the name of the subscription to create, *table_rep_def* is the name of the table replication definition the subscription is for, *function_rep_def* is the name of the function replication definition the subscription is for, *pub* is the name of the publication the subscription is for, *db_repdef* is the name of the database replication definition the subscription is for, and *server_name.db* identifies the primary or replicate database.

The **without materialization** clause activates the subscription without first initializing the subscription data. Use **create subscription** with the **without materialization** clause when there is no activity at the primary database and the data already exists in the replicate database.

define subscription Command

Use **define subscription** to create a subscription with bulk materialization.

Execute the **define subscription** command at the Replication Server that is managing the database where the data is to be replicated. **define subscription** sets the subscription status to DEFINED. The syntax for **define subscription** is:

```
define subscription subscription
for {table_rep_def | function_rep_def
    publication pub_name | database replication definition db_repdef
    with primary at data_server.db
with replicate at data_server.db
[where search_conditions]
[subscribe to truncate table]
```

where *subscription* is the name of the subscription to define, *table_rep_def* is the name of the table replication definition the subscription is for, *function_rep_def* is the name of the function replication definition the subscription is for, *pub_name* is the publication the subscription is for, *db_repdef* is the database replication definition the subscription is for, and *data_server.db* identifies the primary or replicate database.

The *subscription* name must be unique for the replication definition and replicate database.

See *Replication Server Reference Manual > Replication Server Commands > define subscription*.

See also

- *Manage Replicated Tables* on page 303
- *Manage Replicated Functions* on page 379

activate subscription Command

Use **activate subscription** during bulk materialization to start the distribution of updates from the primary to the replicate database for a subscription.

activate subscription sets the subscription status to ACTIVE.

Execute **active subscription** at the Replication Server where you created the subscription using the **define subscription** command. The syntax for **activate subscription** is:

```
activate subscription subscription
  for { table_rep_def | function_rep_def | publication pub_name |
    with primary at data_server.db }
with replicate at data_server.db
[with suspension [at active replicate only]]
```

where *subscription* is the name of the subscription to activate, *table_rep_def* is the name of the table replication definition the subscription is for, *function_rep_def* is the name of the function replication definition the subscription is for, *pub_name* is the publication the subscription is for, and *data_server.db* identifies the primary or replicate database.

Use the **with suspension** clause to suspend the DSI after the subscription status changes to ACTIVE. This prevents the replicate Replication Server from sending updates for the replicated table before the subscription data is loaded. After loading the data 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 table until the subscription is materialized.

If the database is part of a warm standby application, the **with suspension** clause suspends the DSI for the active and standby databases. This let you load the data into both databases before allowing updates to the active database. If you load the data into the active database with logging, use the **with suspension at active replicate only** clause so that the standby DSI remains active. In this case, subscription data is replicated from the active database. The DSI for the active database in a warm standby application is suspended. The clause does not suspend the DSI for the standby database.

See *Replication Server Reference Manual > Replication Server Commands > activate subscription*.

validate subscription Command

Use **validate subscription** to complete the bulk materialization process and set the subscription status to VALID.

Execute **validate subscription** at the Replication Server where you created the subscription. The syntax is:

```
validate subscription subscription
  for { table_ref_def | function_rep_def | publication pub_name
```

```
with primary at data_server.db }
with replicate at data_server.db
```

where *subscription* is the name of the subscription to validate, *table_rep_def* is the name of the table replication definition the subscription is for, *function_rep_def* is the name of the function replication definition the subscription is for, *pub_name* is the publication the subscription is for, and *data_server.db* identifies the primary or replicate database.

check subscription Command

Use **check subscription** to report the status of a subscription at the Replication Server where you enter the command

The subscription status at the primary and replicate Replication Servers often differs while the subscription is being created, so you should enter **check subscription** at both sites. If the primary and replicate databases are managed by a single Replication Server, **check subscription** displays the status of the subscription for both the primary and replicate databases.

The syntax is:

```
check subscription subscription
for { table_rep_def | function_rep_def | publication pub_name |
      database replication definition db_repdef
      with primary at data_server.db }
with replicate at data_server.db
```

where *subscription* is the name of the subscription to check, *table_rep_def* is the name of the table replication definition the subscription is for, *function_rep_def* is the name of the function replication definition the subscription is for, *pub_name* is the publication the subscription is for, *db_repdef* is the database replication definition the subscription is for, and *data_server.db* identifies the primary or replicate database.

The message returned by the command contains subscription status information. If the subscription had an error, the message directs you to the log where you should look for specific error messages.

See *Replication Server Reference Manual > Replication Server Commands > check subscription* for a list of the messages **check subscription** can return.

Direct Load Materialization Progress

The **check subscription** command reports progress information for subscriptions created with the **direct_load** option that are in either the:

- initial-load phase – the period of time Replication Server consumes when it selects rows from the primary database and applies the selected rows to the replicate database
- catchup-up phase – the period of time between the end of the initial load and when the subscription changes to the VALID state

In the initial-load phase for a subscription named *my_sub*, the **check subscription** command reports status similar to:

Manage Subscriptions

```
Subscription my_sub is ACTIVE at the replicate.  
Subscription my_sub is ACTIVE at the primary.  
Subscriptions my_sub progress: initial loading, xx% done, xxxxx  
commands remaining.
```

In the catch-load phase for a subscription named `my_sub`, the `check subscription` command reports status similar to:

```
Subscription my_sub has been MATERIALIZED at the replicate.  
Subscription my_sub is VALID at the primary.  
Subscriptions my_sub progress: catchup, xx% done, xxxxx commands  
remaining.
```

drop subscription Command

Use **drop subscription** to drop subscriptions for either table or function replication definitions.

Dropping a subscription causes Replication Server to stop sending changes from a primary database to a replicate database.

Execute **drop subscription** at the replicate Replication Server. It requires **create object** permission at the replicate Replication Server and **create object** or **primary subscribe** permission at the primary Replication Server.

Here is the syntax:

```
drop subscription subscription  
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.db }  
with replicate at data_server.database  
[without purge  
[with suspension [at active replicate only ] ] |  
[incrementally] with purge]
```

If you choose the **without purge** dematerialization method, Replication Server does not delete subscription data from the replicate database.

If you choose the **with purge** dematerialization method, Replication Server logs in to the replicate database and selects data from it. If this data does not belong to any other subscriptions, the subscription data is deleted from the replicate database.

When you drop subscriptions to table replication definitions, you can purge subscription rows regardless of the materialization method you used when you created the subscription. Rows are removed only if they do not match another subscription.

You can use **check subscription** to view the progress of the **drop subscription**. When the subscription status no longer exists at the primary and replicate Replication Servers, the command is complete.

Subscriptions to function replication definitions are always dropped without purging the replicate data associated with the function. You do not need to specify the **without purge** option.

When you are dropping subscriptions to table replication definitions, you have two basic methods to choose from. Because each method carries important implications, Replication Server requires that you explicitly choose one of these two methods:

- **with purge** – Replication Server removes, or dematerializes, the subscription's rows from the replicate database, if they do not belong in other remaining subscriptions. The Replication Server logs in as the maintenance user to perform the **select** operation. Use the **incrementally** option to specify that dematerialization occurs in 1000-row increments of deletes per transaction.
- **without purge** – the subscription's rows remain at the replicate database. The **with suspension** option leaves the connection to the replicate database suspended when **drop subscription** has completed, so that you can manually remove the rows.

For warm standby applications, the option **with suspension at active replicate only** suspends the active replicate database but not the standby replicate database.

Warning! When removing rows manually, do not remove rows for remaining overlapping subscriptions that require those rows.

Examples:

- To drop a subscription with purge, enter:


```
drop subscription publishers_sub
for publishers_rep
with replicate at SYDNEY_DS.pubs2
with purge
```
- To drop a subscription without purge, enter:


```
drop subscription publishers_sub
for publishers_rep
with replicate at SYDNEY_DS.pubs2
without purge
```
- To drop a subscription without purge and also suspend the DSI for the replicate database so that you can manually delete the rows for the subscription, enter:


```
drop subscription publishers_sub
for publishers_rep
with replicate at SYDNEY_DS.pubs2
without purge
with suspension
```
- If you have a warm standby application for the replicate database, you may want to suspend the connection for the active database only, and leave the standby DSI up. This way, Replication Server will replicate your row deletion transactions from the active replicate database to the standby database. In this case, enter:


```
drop subscription publishers_sub
for publishers_rep
with replicate at SYDNEY_DS.pubs2
```

Manage Subscriptions

```
without purge  
with suspension at active replicate only
```

See *Replication Server Reference Manual* > *Replication Server Commands* > **drop subscription**.

Subscription Example

The subscription example shows you how to use RCL commands to replicate the `publishers` table from a primary Adaptive Server database to a replicate Adaptive Server database by creating an atomic subscription to the table replication definition. You can also do this in Sybase Central.

In the example replication system, at the:

- Primary site:
 - The Replication Server is named `TOKYO_RS`.
 - The primary version of the `publishers` table is in the `pubs2` database of the Adaptive Server named `TOKYO_DS`. You have added a connection from `TOKYO_RS` to the `pubs2` database using Sybase Central or `rs_init` and set up a RepAgent for the database.
 - The system database for `TOKYO_RS` is named `TOKYO_RSSD` and is managed by the `TOKYO_DS` Adaptive Server.
 - A route exists from `TOKYO_RS` to `SYDNEY_RS`.
- Replicate site:
 - The Replication Server is named `SYDNEY_RS`.
 - The replicate copy of the `publishers` table is in the `pubs2` database of the Adaptive Server named `SYDNEY_DS`. You have added a connection from `SYDNEY_RS` to the `pubs2` database using Sybase Central or `rs_init`.
 - The system database for `SYDNEY_RS` is named `SYDNEY_RSSD` and is managed by the `SYDNEY_DS` Adaptive Server.

Replicating Tables in the Example Replication System

Learn the procedures for replicating tables between two Adaptive Servers.

1. To prepare the replicate tables, check replication system components by using Sybase Central or `isql` to log in to the servers identified for the primary and replicate sites.
2. To prepare the primary table in the `TOKYO_DS` primary data server, log in to the `pubs2` database in `TOKYO_DS`, and verify that the `publishers` table exists:

```
isql -Usa -P -STOKYO_DS  
use pubs2  
go
```

```
sp_help publishers
go
```

3. To create subscriptions, prepare login names and grant the relevant permissions for the “pubs2_user” user that is creating the subscription in the TOKYO_DS Adaptive Server. This user must also exist in both Replication Servers:

- a) In TOKYO_DS, create the “pubs2_user” login name:

```
isql -Usa -P -STOKYO_DS
sp_addlogin pubs2_user, pubs2_pw, pubs2
go
```

- b) In TOKYO_DS, add the “pubs2_user” login name to the pubs2 database, and grant pubs2_user **select** permission on the publishers table:

```
use pubs2
go
sp_adduser pubs2_user
go
grant select on publishers to pubs2_user
go
```

- c) In the TOKYO_RS primary Replication Server, create the “pubs2_user” login name and grant **primary subscribe** permission to pubs2_user:

```
isql -Usa -P -STOKYO_RS
create user pubs2_user
set password pubs2_pw
go
grant primary subscribe to pubs2_user
go
```

- d) In the SYDNEY_RS replicate Replication Server, create the “pubs2_user” login name and grant **create object** permission to pubs2_user in the SYDNEY_RS replicate Replication Server:

```
isql -Usa -P -SSYDNEY_RS
create user pubs2_user
set password pubs2_pw
go
grant create object to pubs2_user
go
```

4. In TOKYO_RS, create the replication definition **publishers_rep** for the publishers table:

```
isql -Ujohn -P -STOKYO_RS
create replication definition publishers_rep
with primary at TOKYO_DS.pubs2
with all tables named 'publishers'
(pub_id char(4), pub_name varchar(40),
city varchar(20), state char(2))
primary key (pub_id)
searchable columns (pub_id, pub_name)
replicate minimal columns
go
```

Manage Subscriptions

In this example, the user “john” creates the replication definition. This user requires **create object** permission in TOKYO_RS.

5. In TOKYO_DS, mark the `publishers` primary table for replication. To mark the table for replication with the **sp_setreptable** system procedure, you must be the database owner or system administrator for the data server. Enter:

```
sp_setreptable publishers, 'true'  
go
```

6. In the SYDNEY_DS replicate data server, log in to the pubs2 database, and verify that the `publishers` table exists:

```
isql -Usa -P -SSYDNEY_DS  
use pubs2  
go  
sp_help publishers  
go
```

When you add the replicate `pubs2` database using Sybase Central or **rs_init**, the maintenance user is created and given **replication_role**. The maintenance user must have **replication_role**, **sa_role**, or alias the database owner to replicate **truncate table**.

In SYDNEY_DS, verify that the maintenance user has **select**, **insert**, **delete**, and **update** permissions on the `publishers` table:

```
grant all on publishers to SYDNEY_DS_maint  
go
```

7. Log in to SYDNEY_RS Replication Server as `pubs2_user` and create the subscription **publishers_sub** for the replication definition **publishers_rep**:

```
isql -Upubs2_user -Ppubs2_pw -SSYDNEY_RS  
create subscription publishers_sub  
for publishers_rep  
with replicate at SYDNEY_DS.pubs2  
subscribe to truncate table  
go
```

This subscription uses the default atomic materialization. No **where** clause is included, so all rows will be replicated. Execution of the **truncate table** command will be reproduced at the destination database.

8. While still logged into SYDNEY_RS, use the **check subscription** command to monitor the status of the subscription materialization:

```
check subscription publishers_sub  
for publishers_rep  
with replicate at SYDNEY_DS.pubs2  
go
```

9. You can verify if replication is occurring as expected by verifying that a row you insert is copied to the replicate table.

- a) In TOKYO_DS, insert a row into the `publishers` table:

```
isql -Usa -P -STOKYO_DS  
use pubs2
```



```
go
insert publishers
values ('9950', 'Who Donut', 'Butler', 'CA')
go
```

- b) In SYDNEY_DS, verify that the row you inserted was replicated into the replicate copy of the publishers table:

```
isql -Usa -P -SSYDNEY_DS
use pubs2
go
select * from publishers
go
```

Materialize text, untext, image, and rawobject Data

In general, you can use any materialization method for subscriptions for tables with columns that use the `text`, `untext`, `image`, or `rawobject` datatypes.

If you use atomic or nonatomic materialization, the Replication Server managing the replicate database selects all of the subscription data into a subscription materialization queue.

If you want to materialize `text`, `untext`, `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.

If you are materializing many large data rows, make sure that the Replication Server has sufficient queue space for the data before you create the subscription. For tables with a large volume of `text`, `untext`, `image`, and `rawobject` data, you may need to add temporary partitions to the Replication Server to complete the materialization.

Nonatomic Materialization

If you are using nonatomic subscription materialization and you have set the **replicate_if_changed** replication status for any `text`, `untext`, `image`, or `rawobject` column, Replication Server displays a warning message in the error log file.

You are cautioned that data may be inconsistent if applications modify the primary table during subscription materialization. Run the **rs_subcmp** program to reconcile the data in the replicate and primary tables.

Row Migration

Under certain conditions, `text`, `untext`, `image`, or `rawobject` column data may be missing in a replicate table as a result of row migration.

Row migration occurs in a subscription that has a **where** clause. Updating a column specified in the **where** clause can cause a row to become valid for, or migrate into, the subscription.

When this happens, Replication Server executes an **insert** in the replicate table. To insert a

Manage Subscriptions

complete row, each **insert** would require values for all columns, including `text`, `unitext`, `image`, and `rawobject` columns that did not change in the primary table.

If your application allows rows to migrate into a subscription and you have set any `text`, `unitext`, `image`, or `rawobject` columns to the **replicate_if_changed** replication status, Replication Server displays a warning message in the error log. The message states that a row has migrated into the subscription but that its `text`, `unitext`, `image`, or `rawobject` data is missing.

If a `text`, `unitext`, `image`, or `rawobject` 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`, `image`, or `rawobject` data. Run the **rs_subcmp** program to reconcile the data in the replicate and primary tables.

Subscriptions for Columns with Heterogeneous Datatypes

You create subscriptions for table replication definitions in the normal manner when class-level or column-level translations are defined and active. However, certain restrictions apply to use of the **where** clause.

- Subscriptions that specify columns subject to class- and column-level translations in the **where** clause cannot be dematerialized automatically. You must use the bulk or no-materialization method.
- Take care creating or defining subscriptions that specify class- or column-level translations in the **where** clause. Make sure that the value in the **where** clause comparison is in the declared datatype format. HDS translations take place *after* the subscription is presented.

For example, if searchable column `starttime` is declared as `datetime` but published as `rs_db2_time`, then the comparison value in the **where** clause must be described using `datetime` format.

```
create subscription db2_time_sub
for table_rep_def XXXXX
    with primary at AAAAA
    with replicate at BBBBB
where starttime > '19000101 23:14:02'
```

and not “`where starttime > '23:14:02,'`” which is `rs_db2_time` format.

See the information on heterogeneous datatype translations when you configure replicated tables, and see the *Replication Server Heterogeneous Replication Guide*.

See also

- *Manage Replicated Tables* on page 303

Bitmap Subscriptions

Bitmap subscriptions allow you to create subscriptions that replicate rows based on bitmap comparisons.

When you create a replication definition for a table, specify the datatype of your bitmap columns as `rs_address`. This special datatype tells Replication Server to treat these `int` columns as bitmaps.

The **create subscription** and **define subscription** commands support a bitmap comparison operator (`&`) in the **where** clause for `rs_address` columns or parameters.

In the Adaptive Server table, you use an `int` column to hold a bitmap, since Adaptive Server allows bitwise operators on integer values. An `int` column has 32 bits. You can have multiple `rs_address` columns in a replication definition if your application requires more than 32 bits.

When you create a subscription, specify bitmap comparisons by comparing each `rs_address` column to a bitmask using the `&` operator. Each subscription can have one comparison per `rs_address` column.

Bitmap Subscription Example

For example, consider an application that uses an `rs_address` column named `book_type` to record the categories of books customers are interested in reading.

Table 33. Example Bitmap Comparison

Bit Number	Book Category
0	Science fiction
1	Mystery
2	Business
3	Cooking
4	Popular computing
5	Computer science
6	Psychology
7	Reference

If a bit is set, the customer has expressed interest in books of the corresponding category. The bits are numbered from least significant to most significant. For example, if the customer is interested in mystery, cooking, computer science, and psychology books, the least significant

Manage Subscriptions

8 bits are 01101010 and the 32-bit integer value is 106. The `book_type` column in the customer's row contains the value 106.

To create a subscription for customers who are interested in specified book categories, form a bitmask of the desired categories and compare it, using the `&` operator, to the `book_type` column in the **where** clause of the **create subscription** or **define subscription** command. The `&` operator performs a bitwise AND operation. If the result is non-zero, the row matches the subscription.

For `rs_address` columns 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]
```

For example, to create a subscription for all customers who are interested in mystery or business books, the lower 8 bits of the mask are 00000110. Converted to a 32-bit integer value, the bitmask is 6. For atomic or nonatomic materialization, you can create the subscription as follows:

```
create subscription mystery_or_business  
for customers  
with replicate at BRANCH_22.BOOK_DB  
where book_type & 6
```

You can use a similar approach in the **define subscription** command, used for bulk materialization. For subscriptions to function replication definitions, which require the non-materialization method or bulk materialization, specify parameter names instead of column names.

In addition to 32-bit integer values, you can also compare `rs_address` columns to 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! Hexadecimal values are treated as binary strings by both 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 might not work if a replication system involves different platforms. Be very cautious about comparing `rs_address` columns to hexadecimal numbers in the **where** clause of a subscription.

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 is an optimization for applications that use `rs_address` columns to specify the destination replicate databases.

See *Replication Server Reference Manual > Replication Server Commands* for more information about creating bitmap subscriptions using **create subscription** and **create replication definition**.

Refer to the *Adaptive Server Enterprise Reference Manual* and the *Open Client and Open Server Common Libraries Reference Manual* for more information about conversions between datatypes.

See also

- *Use the where Clause* on page 418

Obtain Subscription Information

Once data is replicating, you may need to obtain information about the subscriptions or verify that data is replicating consistently. Replication Server provides stored procedures for obtaining information and a standalone utility for verifying consistency.

Display Subscription Information

Use the **rs_helpsub** and **rs_helprepdb** stored procedures in the Replication Server RSSD to display information about subscriptions at a Replication Server.

Use **rs_helpsub** to display information about subscriptions at a Replication Server. The syntax is:

```
rs_helpsub [subscription_name
           [, replication_definition
           [, data_server, database]]]
```

Use the **rs_helprepdb** stored procedure to display information about databases with subscriptions for replication definitions in the current Replication Server. The syntax is:

```
rs_helprepdb [, data_server, database]
```

For parameter descriptions, see *Replication Server Reference Manual > RSSD Stored Procedures*.

Verify Subscription Consistency

Learn the types of inconsistency that may occur between primary and replicate tables.

After you create a subscription, Replication Server propagates transactions from the primary database to the replicate database. The replication system keeps the replicate copy of the table consistent with the primary copy.

The replicate data may become inconsistent with the primary version. For example, if you have not restricted update permissions on a replicate table to the maintenance user for the database, a client may update the replicate data directly, introducing inconsistencies.

Manage Subscriptions

Primary and replicate tables may be temporarily inconsistent because Replication Server takes some time to transfer updates from the primary table to the replicate table. However, as soon as the Replication Server applies the updates at the replicate database, inconsistency due to latency no longer exists.

There are three kinds of inconsistency that may occur between primary and replicate tables:

- Missing rows in the primary table are missing from the replicate table.
- Inconsistent rows in the primary table differ from the corresponding rows in the replicate table.
- Orphaned rows in the replicate table do not exist in the primary table or do not match subscriptions for the replicate table.

You need to differentiate between temporary inconsistencies caused by delay and real inconsistencies caused by the incorrect use of the system or by system failures. The **rs_subcmp** program described in the following section helps you make this distinction. You can correct inconsistencies by dropping and recreating subscriptions or by using **rs_subcmp**.

Use rs_subcmp To Locate and Correct Inconsistencies

Use the **rs_subcmp** standalone executable program to compare a replicate table to the primary version of the table, find—and correct if you so choose—missing, orphaned, and inconsistent rows in Sybase databases.

On UNIX systems, the 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. Refer to the Replication Server installation and configuration guides for your platform for more information.

The program works by logging in to the primary data server and the replicate data server, and selecting and comparing rows from both tables.

Because some differences between primary and replicate data can be attributed to latency, **rs_subcmp** first identifies inconsistencies, and then performs iterations a specified number of times. **rs_subcmp** waits for any updates to be replicated before removing the corrected rows from its list.

It is best to use **rs_subcmp** when latency is low to avoid the program having to perform several iterations through the data.

You can instruct **rs_subcmp** to display inconsistent rows on the standard output, correct them, or both display and correct them.

Creating a configuration file avoids the need for complex command lines, which are prone to errors. Here is an **rs_subcmp** configuration file that compares the `sales` table in the `pubs2` database in the data servers `TOKYO_DS` and `SYDNEY_DS`:

```
PDS=TOKYO_DS
RDS=SYDNEY_DS
PDB=pubs2
```

```
RDB=pubs2
RTABLE=sales
RSELECT=select * from sales \
        order by stor_id, ord_num
RUSER=sa
KEY=stor_id
KEY=ord_num
RECONCILE=Y
RECONCILE_CHECK=Y
WAIT=15
NUM_TRIES=5
VISUAL=Y
```

The **PTABLE**, **PSELECT**, and **PUSER** parameters, which are used for the primary database, are not shown in this example. Their values are the same as those of corresponding parameters in the replicate databases, so they need not be included in the configuration file.

The **RSELECT** line and the **PSELECT** line (if used) must be entered on one line. To continue a line onto the next line (row), precede each newline character with a backslash as, for example:

```
RSELECT=select * from sales \
        order by stor_id, ord_num
```

Note: Due to update filtering, columns of `rs_address` datatype may not be identical between the primary and replicate databases. Do not select `rs_address` columns using **RSELECT** or **PSELECT** parameters.

When you execute **rs_subcmp**, you can override values in the configuration file with command line options. For example, rather than changing the name of the TOKYO_DS data server to TOKYO_DS2 in the configuration file, you can specify it on the command line, using the **-S** flag, as the following example illustrates:

```
rs_subcmp -f sales_cmp -S TOKYO_DS2 > sales_badrows
```

In this example, the **-f** option specifies a configuration file name, `sales_cmp`. If the **VISUAL** parameter is set to “Y” in the configuration file (equivalent to the **-V** command line option), a list of the inconsistent rows is generated. In this example, the output is redirected to a file.

Schema Comparison

Schema comparison is useful in comparing schema between two databases that may have the same data but different schemas.

For example, if you want to compare all schemas between two databases using the `config.cfg` file:

```
rs_subcmp -f config.cfg
```

A report file which details the comparison result between two tables or two databases is created after every schema comparison. 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 where you issued the **rs_subcmp**.

Note: Before running **rs_subcmp** for schema comparison, make sure that **ddlgen** is working on your environment.

See *Replication Server Reference Manual > Executable Programs > rs_subcmp*.

Manual Data Reconciliation

To verify the reconciliation of statements before execution, you can create a reconciliation file using **rs_subcmp**.

Use the command line option **-g** with **rs_subcmp** or you can set the configuration file parameter **RECONCILE_FILE** to “Y” to indicate the creation of a reconciliation file.

rs_subcmp Performance Enhancement

Hash algorithm improves the performance of **rs_subcmp** and compresses the data in primary and replicated tables.

The compressed data is then fetched by **rs_subcmp**.

Instead of taking the entire row of data during comparison between the primary table and replicated table, **rs_subcmp** now transfers only the compressed data of each data row from the primary or replicated tables, and then verifies or reconciles inconsistencies between them.

For an improved **rs_subcmp** performance, use the command line parameters **-h** or **-H** or their equivalent configuration file parameters **FASTCMP** or **HASH_OPTION**.

Note: To support hash algorithm, **rs_subcmp** requires ASE 15.0.2 or later and cannot handle case-sensitive comparison. It also cannot handle `text`, `unitext`, or `image` datatypes and does not allow the user to specify the precision for the `float` datatype (maximum precision is used). Also, Sybase suggests to set the ASE parameter **default data cache** to 128M or higher to get a better comparison performance.

The **rs_subcmp** program has a large number of options, which you can specify on the command line or in a configuration file. See *Replication Server Reference Manual > Executable Programs > rs_subcmp* for a list of these configuration file parameters and command line options.

Publication Subscriptions

With publication subscriptions, you create subscriptions for a group of replication definitions using a single command.

You collect replication definitions and their articles in a publication at the primary Replication Server. At the replicate Replication Server, you create a publication subscription against that publication.

When you create a publication subscription, Replication Server creates a subscription for each article in the publication.

Publication subscriptions and article subscriptions follow the rules and requirements for single subscriptions with one exception: They cannot contain **where** clauses. To specify a subset of rows that the replicate Replication Server receives, include **where** clauses in the article.

To use publications, the primary Replication Server must be version 11.5 or later. To use publication subscriptions, the replicate Replication Server and the route from the primary Replication Server and the replicate Replication Server must be version 11.5 or later.

The following restrictions apply:

- A valid publication must exist before you can create a publication subscription against it.
- The name of a publication subscription must be unique to the publication, to the destination data server, and to the destination database.
- You can include articles in one or more publications that reference different replication definitions for the same primary table. However, you cannot subscribe to more than one replication definition per primary table for each replicate table.

Use the command line to create and manage publication subscriptions.

See also

- *Specifying a where Clause with the create article Command* on page 368
- *Use Publications to Replicate Data at the Command Line* on page 365
- *Use Publications* on page 364

Commands for Creating and Managing Publication Subscriptions

Learn the commands you can use to create and manage publication subscriptions.

All of the commands, except **check subscription**, require **primary subscribe** or **create object** permission at the source Replication Server and **create object** permission at the destination Replication Server. Anyone can execute **check subscription**.

Table 34. Commands for Managing Publication Subscriptions

Command	Task
create subscription <i>sub_name</i> for publication <i>pub_name</i>	Creates a subscription for a publication and a subscription for each article in the publication. With create subscription , you can: <ul style="list-style-type: none"> • Subscribe to table replication definitions using the atomic, non-atomic, or no-materialization method. • Subscribe to function replication definitions using the no-materialization method.

Command	Task
define subscription <i>sub_name</i> for publication <i>pub_name</i>	Defines a subscription for a publication and a subscription for each article in the publication. Use with activate subscription and validate subscription . With define subscription , you can subscribe to articles with table replication definitions or function replication definitions using the bulk materialization method.
activate subscription <i>sub_name</i> for publication <i>pub_name</i>	Activates a subscription for a publication and a subscription for each article in the publication. Use with define subscription and validate subscription for bulk materialization.
validate subscription <i>sub_name</i> for publication <i>pub_name</i>	Validates a subscription for a publication and a subscription for each article in the publication. Use with define subscription and activate subscription for bulk materialization.
check subscription <i>sub_name</i> for publication <i>pub_name</i>	Displays the status of the publication subscription and all of its article subscriptions.
check subscription <i>sub_name</i> for article <i>article_name</i> in <i>pub_name</i>	Displays the materialization status of an article subscription.
rs_helppubsub	Displays information about publication subscriptions.
drop subscription <i>sub_name</i> for publication <i>pub_name</i>	Removes the publication subscription and all of its article subscription from the <code>rs_subscriptions</code> system table at the primary and replicate sites.
drop subscription <i>sub_name</i> for article <i>article_name</i> in <i>pub_name</i>	Removes the article subscription from the publication subscription and from the <code>rs_subscriptions</code> system table at the primary and replicate sites.

See also

- *Use the create subscription Command to Create a Publication Subscription* on page 441
- *Create Publication Subscriptions with Bulk Materialization* on page 443
- *Display Subscription Information* on page 435
- *Drop Subscriptions for Publications and Articles* on page 445
- *Commands for Creating and Managing Publications* on page 365

Enable Replication of the truncate table Command

You can enable replication of **truncate table** to the replicate table when you create, refresh, or define a publication subscription.

If you do not enable replication of **truncate table** to the replicate table, you must execute **truncate table** yourself at the replicate database.

For example, to create the publication subscription **pubs2_sub** and enable replication of **truncate table**, at the destination Replication Server enter:

```
create subscription pubs2_sub
  for publication pubs2_sub
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
  subscribe to truncate table
```

All subscriptions to the same replicate table must use **truncate table** consistently. If a replicate table has a subscription that does not enable replication of **truncate table** and you add another subscription that does enable replication of **truncate table**, the publication subscription fails.

You do not need to include **subscribe to truncate table** when you activate and validate the publication subscription.

See also

- *Enable Replication of truncate table* on page 419

Create Publication Subscriptions

Once a publication has been validated, you can create subscriptions against it.

When you create a publication subscription, Replication Server creates a subscription for each article in the publication.

Publication subscriptions and article subscriptions specify the publication, the primary and replicate databases, and the materialization method. They do not contain **where** clauses. To specify a subset of rows to be replicated, include **where** clauses in the article description.

See also

- *Specifying a where Clause with the create article Command* on page 368

Use the create subscription Command to Create a Publication Subscription

Use **create subscription** to create a publication subscription and an article subscription for each article in the publication.

You can use **create subscription** to materialize source data at the destination database using the atomic, nonatomic, or no-materialization method.

Execute **create subscription** at the Replication Server that manages the destination database. Subscription information is stored in the `rs_subscriptions` system tables at the primary and replicate sites.

This example creates a subscription named **pubs2_sub** for the publication **pubs2_pub**. It also creates a subscription named **pubs2_sub** for each article in **pubs2_pub**. The source database is `pubs2` managed by the `TOKYO_DS` data server. The destination database is also named `pubs2`; it is managed by the `SYDNEY_DS` data server.

```
create subscription pubs2_sub
  for publication
```

```
with primary at TOKYO_DS.pubs2  
with replicate at SYDNEY_DS.pubs2
```

Note: The maximum size of a **where** clause in a **create subscription** statement is 255 characters.

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

Specify a Materialization Method for Publication Subscriptions

Specify materialization methods for publication subscriptions in the same way you specify materialization methods for regular subscriptions.

When you use **create subscription**, you can specify atomic, nonatomic, or the no-materialization method. The default method is atomic materialization, using the **select with holdlock** operation.

Article subscriptions share the name of the parent subscription and generally, its materialization method. However, function replication definitions require the bulk or no-materialization method. If you use **create subscription**, and articles in the publication reference function replication definitions, Replication Server uses the no-materialization method for these article subscriptions—regardless of the materialization method specified in the publication subscription.

See also

- *Subscription Materialization Methods* on page 398

Refresh Publication Subscriptions

When you add articles to an existing publication, you must add article subscriptions to the existing publication subscription to subscribe to the new articles. Use **for new articles** to refresh the subscription.

This clause instructs Replication Server to check the subscription against the publication and then to create subscriptions for any unsubscribed articles.

For example, to refresh the publication subscription **pubs2_sub**, enter this command at the destination Replication Server:

```
create subscription sub for publication pub  
with primary at TOKYO_DS.pubs2  
with replicate at SYDNEY_DS.pubs2  
for new articles
```

Use **check subscription** to find out whether a subscription exists for each article in a publication.

See also

- *View Publication Subscription Information* on page 445

Create Publication Subscriptions with Bulk Materialization

Bulk materialization allows you to load subscription data from media such as magnetic tape. Use this method if the amount of data to be transferred is too large to copy through the network.

You can also use this method to create subscriptions for function replication definitions.

When you create publication subscriptions with bulk materialization, you must use **define subscription**, **activate subscription**, and **validate subscription**. You use these bulk materialization commands to create publication subscriptions in the same way you create single subscriptions. You cannot include **where** clauses in publication subscriptions.

See also

- *Specifying a where Clause with the create article Command* on page 368

Use the define subscription Command for Publication Subscriptions

Use **define subscription** to create a publication subscription and a subscription for each article in the publication.

define subscription always creates a subscription using bulk materialization. Execute **define subscription** at the Replication Server that manages the destination database. Subscription information is stored in the `rs_subscriptions` system tables at the source and destination sites.

All subscriptions in the publication subscription are created at the same time.

This example 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
```

When you define a publication subscription with bulk materialization, you can enable replication of **truncate table** to the destination table.

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

See also

- *Enable Replication of the truncate table Command* on page 440

Use the activate subscription Command for Publication Subscriptions

Use **activate subscription** to activate a publication subscription and its subscription subset.

Execute **activate subscription** at the Replication Server that manages the destination database.

Manage Subscriptions

Before you execute **activate subscription**, you must execute **define subscription**, and the publication subscription status must be DEFINED.

All subscriptions in the publication subscription are activated at the same time.

For example, to activate every subscription in the **pubs2_sub** publication subscription, enter:

```
activate subscription sub for publication pub
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

Use the validate subscription Command for Publication Subscriptions

Use **validate subscription** to set the subscription status to VALID for the publication subscription, and its subscription subset.

Execute **validate subscription** at the Replication Server that manages the replicate database.

Before you execute **validate subscription**, you must execute **activate subscription** and the publication subscription status must be ACTIVE.

All subscriptions in the publication subscription are validated at the same time.

The following example validates every subscription in the publication subscription **pubs2_sub**.

```
validate subscription sub for publication pub
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
```

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

Refresh Publication Subscriptions Using Bulk Materialization

When you refresh a publication subscription using bulk materialization, use the **for new articles** clause when you define the publication subscription.

You do not need to repeat the clause when you activate and validate the subscription. The following example refreshes the publication subscription **pubs2_sub**.

```
define subscription pubs2_sub
  for publication pubs2_pub
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
  for new articles
```

See *Replication Server Reference Manual > Replication Server Commands > define subscription* for syntax and usage guidelines.

To check whether a subscription exists for each article in a publication, execute **check subscription** at the primary or replicate Replication Server.

See also

- *Check Publication and Article Subscription Status* on page 446

Drop Subscriptions for Publications and Articles

Use **drop subscription** to drop a publication subscription and all of its article subscriptions, or to drop a single article subscription.

drop subscription removes information about the publication subscription and its article subscriptions from system tables at the source and destination servers. It does not remove publication information from the destination server. Thus, you can create another subscription against the publication, and Replication Server only needs to reload primary site information if it has been changed.

Include the **without purge** clause to retain existing rows replicated by the subscription to the destination database. The subscriptions are dropped all at once.

This example drops a subscription named **pubs2_sub** for the publication **pubs2_pub** using **without purge**.

```
drop subscription pubs2_sub
  for publication pubs2_pub
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
  without purge
```

Include the **with purge** clause to delete existing rows replicated by the subscription to the destination database. The subscriptions are dropped one at a time.

This example uses **with purge**:

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

For example, to delete the **pubs2_art** article, without removing rows replicated by the subscription, enter:

```
drop subscription sub for article pubs2_art
  with primary at TOKYO_DS.pubs2
  with replicate at SYDNEY_DS.pubs2
  without purge
```

See *Replication Server Reference Manual > Replication Server Commands* for complete syntax and usage guidelines.

View Publication Subscription Information

You can view information about publication and article subscriptions with the **check subscription** command or the **rs_helppubsub** stored procedure.

Check Publication and Article Subscription Status

Use **check subscription** at the primary Replication Server or the replicate Replication Server to check the status of a publication subscription and its article subscriptions or to check the status of an article subscription.

check subscription returns a status (such as **VALID**, **MATERIALIZING**, or **ACTIVE**) along with a descriptive message. See *Replication Server Reference Manual > Replication Server Commands > check subscription* for complete syntax and usage guidelines and a list of status messages.

- This example displays the subscription status of the publication subscription **pubs2_sub**.

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

If the publication subscription is valid, Replication Server also checks whether the subscription is current. When you alter a publication after the subscription is created, the publication subscription is out of sync with the publication. To create subscriptions for new articles and make the subscription current, refresh the subscription using **create subscription** or **define subscription**.

- This example displays the subscription status of the article **pubs2_art** in the subscription **pubs2_sub**.

```
check subscription sub for article pubs2_art
      in pubs2_pub
      with primary at TOKYO_DS.pubs2
      with replicate at SYDNEY_DS.pubs2
```

Display Publication and Article Subscription Information

To display information about a publication subscription and article subscriptions, use the **rs_helppubsub** stored procedure at either the primary or replicate Replication Server RSSD.

For example:

- To list all publication subscriptions at a site, enter:

```
rs_helppubsub
```

For each publication subscription known to the site, the display includes the names of the subscription and its associated publication, the names of the primary and replicate databases and data servers, status information, and the date of the latest change to the publication subscription.

- To display information about a particular publication subscription, enter:

```
rs_helppubsub subscription_name
```

The output displays the information described in the above example for all publication subscriptions named *subscription_name*.

- To display information about a particular publication subscription and its article subscriptions, enter:

```
rs_helppub subscription_name, publication_name,  
           primary_dataserver, primary_db,  
           replicate_dataserver, replicate_db
```

The output displays the information described in the above examples for all publication subscriptions named *subscription_name*. For each article subscription, the output displays subscription and article name, status information for the primary and replicate Replication Servers, replication definition name, autocorrection status, and the date of the latest change to the article subscription.

See the *Replication Server Reference Manual > RSSD Stored Procedures > rs_helppubsub* for complete syntax and usage guidelines, and sample output.

Manage Replicated Objects Using Multisite Availability

Multisite availability (MSA) can make the process of setting up a replication system both faster and easier.

Some of the features that MSA provides are:

- A simple replication methodology that requires only one replication definition for the primary database and only one subscription for each subscribing database.
- A replication filtering strategy that lets you choose whether or not to replicate individual tables, transactions, functions, system stored procedures, and data definition language (DDL).
- Replication of DDL to any replicate database—including non-warm standby databases.
- Replication to multiple replicate sites—for warm standby as well as non-warm standby databases.

You can overlay MSA scenarios over your existing replication structure. The procedures for implementing MSA are similar to those you use to replicate to warm standby or replicate databases.

Database Replication

When you use table and function replication, you describe each piece of data that is to be replicated using individual table and function replication definitions and subscriptions.

This methodology allows you to transform data and provides fine-grained control over the information being entered in the replicate database. However, it requires that you mark each table or function to be replicated, create a replication definition for each replicated table or function, and create subscriptions for each replication definition at each replicate database.

MSA lets you identify specific database objects: tables, functions, transactions, DDL, and system stored procedures in a single replication definition. You can choose to replicate the entire database, or you can choose to replicate—or not replicate—particular tables, functions, transactions, DDL, and system stored procedures in that database. If you do not need to replicate partial tables, MSA can provide replication while affording the advantages of simple setup and maintenance.

When the Replicate Is a Warm Standby Database

In the non-MSA warm standby scenario, changes to the primary database are copied directly to the warm standby database without alteration.

This methodology allows replication of DDL. To change or qualify the data sent, you must add table and function replication definitions. Each primary database can have one, and only one, standby database. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications* for a complete discussion of this warm standby application.

Manage Replicated Objects Using Multisite Availability

MSA provides all the features of the warm standby application described in *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications*. In addition, MSA:

- Enables replication to multiple standby databases
- Allows you to replicate or not replicate specific database objects

Bidirectional Replication Support for DDL in MSA

You can configure multisite availability (MSA) to set up a two-way replication of DDL transactions between two Adaptive Server databases.

Replication Server 15.0 and later supports this bidirectional replication using a Replication Server configuration parameter called **dsi_replication_ddl**. 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 transactions replication in bidirectional MSA replication environment.

MSA Mixed-Version Environment

In an MSA mixed-environment, the primary Replication Server filters the data features with higher versions.

Incompatible commands are not sent to the standby Replication Server. The configuration parameter **dist_stop_unsupported_cmd** suspends the DIST if there are incompatible commands. You can configure this parameter using one of the following syntaxL

- ```
configure replication server
 set 'dist_stop_unsupported_cmd' to ['on' | 'off']
```
- ```
alter connection srv.db
  set 'dist_stop_unsupported_cmd' to ['on' | 'off']
```
- ```
alter logical connection lsrv.ldb
 set 'dist_stop_unsupported_cmd' to ['on' | 'off']
```

By default, **dist\_stop\_unsupported\_cmd** is off. When the parameter is on, the DIST suspends itself if a command cannot be sent to some destinations. Resume DIST by skipping the entire transaction, or reset the parameter to off.

---

**Note:** To create a database subscription from a primary Replication Server with version 15.0, the replicate Replication Server must be version 15.0 or later.

---

### **See also**

- *Manage Replicated Tables* on page 303
- *Manage Replicated Functions* on page 379

## Setting Up Bidirectional Replication Support for DDL in MSA

---

Learn to set up bidirectional replication support for MSA.

1. Create a bidirectional MSA replication environment.
2. Grant “set session authorization” privilege to the maintenance user on the destination database, as shown in the following example:

```
grant set session authorization to maint_user
```

3. Alter the connection of the destination database to set **dsi\_replication\_ddl** configuration parameter to “on” to enable bidirectional DDL replication, as shown in the following example:

```
alter connection to dataserver.database set dsi_replication_ddl on
```

4. Replicate DDL transactions.

### See also

- *Creating a Bidirectional Replication Environment* on page 74

## Set Up an MSA System

---

You can set up MSA replication using different ways.

Set up representative MSA replication architectures:

- Simple, full-database replication using the default single-path replication or multi-path replication.
- Replication of specified tables and functions
- Replication to multiple replicate databases

To replicate DDL or system stored procedures, you can easily add syntax when you are setting up these architectures.

### See also

- *Replicating DDL and System Procedures* on page 473

## **Replicating the Database in a Simple Scenario**

Use database replication definitions and subscriptions to replicate the entire primary database to one or more replicate databases in this simple scenario.

To configure multi-path replication, see *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Creating Multiple Replication Paths for MSA Environments*.

You can easily add syntax to this example to replicate DDL or system stored procedures.

1. Mark the primary database for replication using **sp\_reptostandby**. For example:

```
sp_reptostandby primary_db, 'all'
```

---

**Note:** **sp\_reptostandby** does not mark user stored procedures for replication. You must mark each user stored procedure individually using **sp\_setrepproc**.

---

See the *Replication Server Heterogeneous Replication Guide* for non-ASE data servers.

2. Set the RepAgent parameter **send warm standby xacts** to true so that RepAgent sends system transactions and DDL to both standby and replicate databases. For example, at the primary data server, enter:

```
sp_config_rep_agent primary_db,
 'send warm standby xacts', 'true'
```

See the *Replication Server Heterogeneous Replication Guide* for non-ASE data servers.

3. Create a database replication definition using **create database replication definition** at the primary Replication Server. For example:

```
create database replication definition repdef_1
 with primary at PDS.primary_db
```

See *Replication Server Reference Manual > Replication Server Commands > create database replication definition* for complete syntax and usage information.

4. Create a database subscription for each subscribing database. In this example, we are creating a database subscription using **create subscription** and the no materialization method. The primary and replicate databases have been synchronized prior to subscription. You can also use **create subscription** if activities at the primary database can be suspended.

For example, at the replicate Replication Server, enter:

```
create subscription sub_1
 for database replication definition repdef_1
 with primary at PDS.primary_db
 with replicate at RDS.rdb
 without materialization
 subscribe to truncate table
```

When creating a database subscription, you can use the no materialization method (as shown in step 4) or the bulk materialization method to synchronize databases. The

procedure you use depends on which materialization method you choose and whether primary table activities can be suspended.

### See also

- *Materialization* on page 469
- *Replicating DDL and System Procedures* on page 473

## Replicating Tables and Functions

Use MSA capabilities to replicate particular tables or functions.

You can easily add syntax to this example to replicate DDL or system stored procedures.

1. Mark tables, stored procedures, and database for replication and create the database replication definition.

In this example, we are replicating `table1` and `table2` only. You can identify particular tables in either of two ways:

- Mark the database for replication using **sp\_reptostandby**. Create the database replication definition and identify specific tables for replication using **create replication definition**. You must also tell the RepAgent to send replicate data to replicate as well as standby databases.

At the primary data server, enter:

```
sp_reptostandby primary_db, 'all'
sp_config_rep_agent primary_db,
'send warm standby xacts', 'true'
```

At the primary Replication Server, enter:

```
create database replication definition rep_1B
with primary at PDS.pdb
replicate tables in (table1, table2)
```

- Alternatively, mark particular tables and stored procedures for replication using **sp\_setreptable** and **sp\_setrepproc**. Then, create the database replication definition. For example:

```
sp_setreptable table1, 'true'
```

```
sp_setrptable table2, 'true'
```

```
create database replication definition rep_1A
with primary at PDS.pdb
```

---

**Note:** You can replicate DDL changes only if you mark its database for replication using **sp\_reptostandby**.

---

See the *Replication Server Heterogeneous Replication Guide* for non-ASE data servers.

2. Create the database subscription. To subscribe without materialization, follow the procedure in the simple scenario to replicate the database. You can also subscribe using bulk materialization.

---

**Note:** You can also use `sp_reptostandby` to mark the database and then create table replication definitions and subscriptions—without creating a database replication definition. This method eliminates the need to mark individual tables, yet allows you to select and replicate partial tables. The database connection parameter `rep_as_standby` must be **on**.

---

Be aware of considerations when dealing with encrypted columns.

### See also

- *Replicate Encrypted Columns* on page 344
- *Materialization* on page 469
- *Replicating the Database in a Simple Scenario* on page 452
- *Replicating DDL and System Procedures* on page 473

## Using Replicate Databases as Warm Standby Databases

Use MSA to replicate DDL and other database objects to multiple replicate or warm standby databases.

You can create database replication definitions and database subscriptions to logical connections. See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications* for detailed information about setting up logical connections.

In this example of a basic setup for a multiple warm standby architecture, you replicate from one primary database—`dsA.db`, to two replicate databases—`dsB.db` and `dsC.db`. There is a single Replication Server controlling replication, and only standby replication takes place to and from the primary database. Only `dsA` can replicate DDL and system stored procedures. If users are switched to `dsB.db` or `dsC.db`, DDL and system stored procedures are not replicated.

You can easily add syntax to this example to replicate DDL or system stored procedures.

---

**Note:** This example uses a different database replication definition for each subscribing site. You could also create a single database replication definition that handles the common set of replicated tables and functions, and then create table and function subscriptions for the tables and functions that are not common to both standby databases.

---

1. Suspend all database activities.
2. Mark `dsA.db`, `dsB.db`, and `dsC.db` for replication using `sp_reptostandby`.
3. At each data server, set **send warm standby xacts** to true for each RepAgent. For example:

```
sp_config_rep_agent dbname,
 'send warm standby xacts', 'true'
```

4. At Replication Server, set **dsi\_replication** off for each connection. For example:

```
alter connection to dsB.db
 set dsi_replication 'off'
```



---

**Note:** Sybase recommends that you set **dsi\_replication** to off for warm standby connections as it prevents replicated data in the transaction log from being replicated again in the event of a switchover. **dsi\_replication** should be turned on (the default) for normal replication.

---

5. Create a database replication definition for each database, defining each as the primary. For example

```
create database replication definition rep_2
 with primary at dsA.db
 replicate DDL
```

```
 replicate system procedures
```

```
create database replication definition rep_2
 with primary at dsB.db
```

```
create database replication definition rep_2
 with primary at dsC.db
```

6. As each database can be a primary or a standby database, create or define subscriptions so that each database subscribes to every other database. You can use different materialization methods for each subscription. For example:

```
create subscription sub_2B
 for database replication definition rep_2
 with primary at dsB.db
 with replicate at dsA.db
 without materialization
 subscribe to truncate table
```

```
create subscription sub_2C
 for database replication definition rep_2
 with primary at dsC.db
 with replicate at dsA.db
 without materialization
 subscribe to truncate table
```

```
define subscription sub_2A
 for database replication definition rep_2
 with primary at dsA.db
 with replicate at dsB.db
 subscribe to truncate table
 use dump marker
```

```
create subscription sub_2C
 for database replication definition rep_2
 with primary at dsC.db
 with replicate at dsB.db
 without materialization
 subscribe to truncate table
```

```
define subscription sub_2A
 for database replication definition rep_2
 with primary at dsA.db
 with replicate at dsC.db
```

## Manage Replicated Objects Using Multisite Availability

```
subscribe to truncate table
use dump marker

create subscription sub_2B
for database replication definition rep_2
with primary at dsB.db
with replicate at dsC.db
without materialization
subscribe to truncate table
```

7. Dump dsA.db.
8. With the dsB.db DSI suspended, load database to dsB.db.
9. Resume connection to dsB.db.
10. With the dsC.db DSI suspended, load database to dsC.db.
11. Resume connection to dsC.db.
12. Resume database activities.

### See also

- *Replicating DDL and System Procedures* on page 473

### Switchover

In any standby situation, switchover involves disconnecting users from the active database and reconnecting them to the new active database. In this case, switchover must wait for the queues to empty so that no transactions are lost.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications* for more information about logical connections and switchover.

## Mark Data for Replication

---

You can mark databases, tables, and functions for replication in MSA using **sp\_reptostandby**, **sp\_setreptable**, and **sp\_setrepproc**.

---

**Note:** See the *Replication Server Heterogeneous Replication Guide* for non-ASE data servers.

---

When the database is marked by **sp\_reptostandby**:

- The RepAgent configuration parameter **send warm standby xacts** must be true.
- User-defined stored procedures are not replicated unless they are marked individually using **sp\_setrepproc**.
- RepAgent sends DDL, system procedures, and transactions to the Replication Server. A database replication definition can filter them out at the Replication Server.
- And you use table replication definitions and table subscriptions, you can send table data to both replicate databases and warm standby databases by setting the database connection parameter **rep\_as\_standby** on.

When the database is not marked by **sp\_reptostandby**, DDL is not replicated for the marked tables and functions. See the Data Replication table for a summary of how data is replicated using the different system procedures for marking data for replication.

The table summarizes how data is replicated.

**Table 35. Data Replication**

| Data Marked By                                                            | Table and Function Subscriptions Only                                                                       | Database Subscription Only                                                                             | Table, Function, and Database Subscriptions Coexist                                                             |
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>sp_setreptable</b> and <b>sp_setrepproc</b>                            | <ul style="list-style-type: none"> <li>Replicate marked data</li> <li>Do not replicate DDL</li> </ul>       | <ul style="list-style-type: none"> <li>Replicate marked data</li> <li>Do not replicate DDL</li> </ul>  | <ul style="list-style-type: none"> <li>Replicate marked data</li> <li>Do not replicate DDL</li> </ul>           |
| <b>sp_reptostandby</b>                                                    | <ul style="list-style-type: none"> <li>Check <b>rep_as_standby</b></li> <li>Do not replicate DDL</li> </ul> | <ul style="list-style-type: none"> <li>Replicate all data</li> <li>Replicate DDL (optional)</li> </ul> | <ul style="list-style-type: none"> <li>Check <b>rep_as_standby</b></li> <li>Replicate DDL (optional)</li> </ul> |
| <b>sp_setreptable</b> , <b>sp_setrepproc</b> , and <b>sp_reptostandby</b> | <ul style="list-style-type: none"> <li>Check <b>rep_as_standby</b></li> <li>Do not replicate DDL</li> </ul> | <ul style="list-style-type: none"> <li>Replicate all data</li> <li>Replicate DDL (optional)</li> </ul> | <ul style="list-style-type: none"> <li>Check <b>rep_as_standby</b></li> <li>Replicate DDL (optional)</li> </ul> |

## Manage Database Replication Definitions

A table, a function, a transaction, DDL, or a system stored procedure can be a replicated object. A database replication definition specifies filters for replicated objects—either including or excluding the same or entire replicated object from replication.

For example, to create a database replication definition:

```
create database replication definition rep_1C
with primary at PDS.pdb
 replicate tables in (table1, table2)
 not replicate functions in (fc_a)
 not replicate system procedures
 replicate transactions
 replicate DDL
```

In this example, we are replicating:

- table1 and table2
- All functions except **fc\_a**
- All transactions

- Supported DDL commands

We are not replicating:

- Any database tables except `table1` and `table2`
- Function **fc\_a**
- Any system procedures

See *Replication Server Reference Manual* > **create database replication definition** for complete syntax and usage information.

---

**Note:** Database replication definitions do not support the options **send-standby-all-columns**, **send-standby-all-parameters**, and **send\_standby\_repdef\_cols**. Where a database replication definition exists, Replication Server sends all columns or parameters.

---

## Altering Database Replication Definitions

Use **alter database replication definition** to change a database replication definition.

**alter database replication definition** allows you to replace one filter at a time. For example:

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

See the *Replication Server Reference Manual* for complete syntax and usage information.

When you execute **alter database replication definition**, Replication Server writes an **rs\_marker** to the inbound queue. The command does not take effect until the marker reaches the Distributor (DIST), which will by then have rebuilt the Database Subscription Resolution Engine (DSRE) to incorporate the changes.

Altering a database replication definition with associated subscriptions may desynchronize replicate tables. To resynchronize, you can either:

- Quiesce Replication Server, drain the transaction log, and apply changes manually, or
- Use the **with\_dsi\_suspended** option, which causes the replicate Replication Server to suspend the replicate DSI when it reads the “alter database replication definition” marker.

To alter a database replication definition and resynchronize replicate tables:

1. Execute **alter database replication definition** and include the **with\_dsi\_suspended** phrase.
2. Wait for the replicate DSI to suspend.
3. Use bulk materialization to resynchronize replicate tables.
4. Resume the connection.

## Drop Database Replication Definitions

You can drop a database replication definition, but you must first have dropped all associated database subscriptions.

See *Replication Server Reference Manual* > **drop database replication definition** for syntax and usage information.

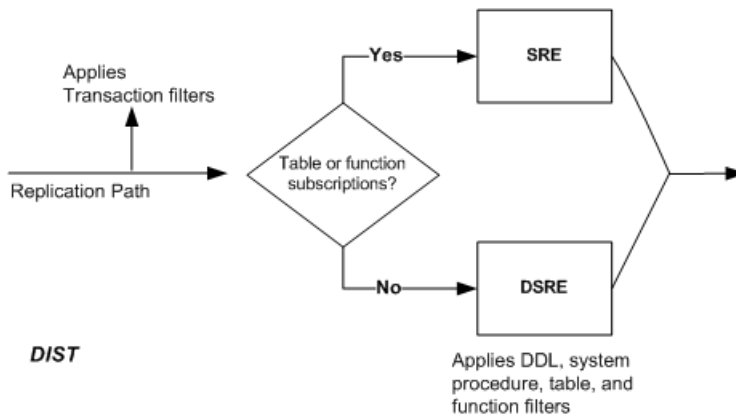
## Database Replication Filters

Learn how replication filters work.

The Subscription Resolution Engine (SRE) evaluates table and function subscription rows. The Database Subscription Resolution Engine (DSRE) evaluates database objects—except transactions. When a database replication definition causes a transaction to be sent to the replicate Replication Server, the DIST evaluates the transaction before other database objects are evaluated by the DSRE and before transaction rows are evaluated by the SRE. Thus, Replication Server filters out transactions even if they contain data that satisfies a database subscription or a table or function subscription.

If a database subscription and a table or function subscription coexist for the same table or function, the table or function subscription takes precedence. In this instance, the DIST does not pass the replicated table or function to the DSRE for evaluation; the DIST passes it to the SRE.

**Figure 20: Evaluation of Database Replication Filters**



## View Information About Database Replication Definitions

---

Use **rs\_helpdbrep** to view information about a specific database replication definition or all database replication definitions for a database or a data server.

For example, to view information about the **rep\_1B** database replication definition, enter:

```
rs_helpdbrep rep_1B, PDS, pdb
```

See *Replication Server Reference Manual > RSSD Stored Procedures > rs\_helpdbrep* for syntax and usage information.

## Concurrently Using Database, Table, and Function Replication Definitions

---

You do not need to add table and function replication definitions when you use a database replication definition. However, to transform the data if a table needs to have a different **replicate minimal columns** or dynamic SQL setting than the target connection, you must do so.

Create and use table or function replication definitions that include the **send standby** clause to:

- Change the name of a replicated table or function
- Change the name of a replicated column
- Publish different column datatypes
- Replicate fewer columns or parameters
- Replicate minimal columns
- Not replicate dynamic SQL

Create and use table or function replication definitions where the **send standby** clause is optional to:

- Declare different table column datatypes
- Set autocorrection for materialization or dematerialization
- Use table or function subscription to override database replication
- Exclude dynamic SQL in the standby database combined with **dynamic\_sql** setting

---

**Note:** As long as a database replication definition and a database subscription are in place, you can use table or function replication definitions without table or function subscriptions. You need to use table or function subscriptions only if you require the functionality they provide.

---

If a database replication definition and table replication definitions exist at the primary, and a database subscription but no table subscriptions exist at the replicate, replication behavior

depends in part on the presence or absence of the **send standby replication definition columns/parameters** clause in the table or function replication definition.

- If the **send standby** clause is present, the database subscription honors the table or function replication definition; the table replication definition's primary key columns and replicate minimal columns settings are used to replicate into the replicate database. The database subscription always treats **send standby all columns** as **send standby replication definition columns**.
- If the table replication definition does not contain the **send standby** clause and other replication definitions exist for a given table, the database subscription replicates data using the internal table replication definition (the union of all such replication definitions). All columns are replicated, and data is converted to the declared columns or datatypes.
- If a table replication definition is made for a table whose owner is not *dbo*, the table must be individually marked to replicate the owner information, and the owner information must be included in the replication definition.

### See also

- *Concurrently Using Database, Table, and Function Subscriptions* on page 471

## Alter Database Replication Definitions

Adding or dropping database replication definitions does not affect table or function subscriptions

When you alter a database replication definition, you replace the existing database replication filter or add the filter category to the database replication filters, if it is a new category.

Replication Server places a marker in the inbound queue and allows the DIST to process the command. The new filter is applied to transactions committed after the marker.

- If a table subscriptions exists, no action is required.
- If no table subscription exists, you must include the **dsi\_suspended** clause in the **alter database replication definition** command, or manually materialize or dematerialize the table.

To alter table and function replication definitions, use the replication definition change request process where Replication Server coordinates the propagation of replication definition changes and data replication automatically.

### See also

- *Replication Definition Change Request Process* on page 350

## Reduce the Use of Replication Definitions and Subscriptions

---

In a replication system containing only Adaptive Server databases, you can reduce the need for replication definitions and subscriptions for tables and stored procedures in a warm standby or multisite availability (MSA) environment.

You need not create a replication definition for a primary table or stored procedure if the sole purpose of the replication definition is to specify some or all of the following:

- Primary key columns
- Table or column names that may be quoted.
- Customized function strings for replicate tables or stored procedures.

### See also

- *Quoted Identifiers* on page 312

## Primary Key Columns and Quoted Table or Column Names

---

RepAgent for Adaptive Server uses Log Transfer Language (LTL) to specify a table or column name that may be quoted, and whether a table column is part of the table primary key. Since RepAgent sends the primary key and quoted identifier information to Replication Server, you do not need a replication definition if the sole purpose of the replication definition is to specify primary key and quoted identifier information, in a replication system that contains only Adaptive Server databases.

The reduced requirement for replication definitions makes it easier to manage a replication environment involving databases with many tables, tables with many columns, or tables with frequent schema changes. Replication performance improves for tables that are currently without replication definitions, as RepAgent directly provides Replication Server with table primary-key information, so that Replication Server packs only the primary-key columns in the **where** clauses of **update**, **delete**, and **select** commands.

- When a replication definition is subscribed by an MSA database, its primary key columns are used for the **where** clause packing for the database.
- When a replication definition is marked **send standby replication definition columns** or **send standby all columns**, primary-key columns of the replication definition are used for the **where** clause packing for the standby and the MSA databases that do not have a subscription to any of the replication definitions of the same table.
- When there are one or more replication definitions for a table, and none of them are marked **send standby**, Replication Server uses the union of the primary key columns for the replication definitions for the **where** clause packing for the standby and MSA databases that do not have subscriptions to any of the replication definitions for the table.



- When there is no replication definition for a table, and LTL sends information about the primary key, the primary key columns indicated by the LTL are used for the **where** clause packing.
- When there is no replication definition for a table, and an LTL command does not indicate any primary column, all columns except `text`, `image`, and `unitext` columns are used for the **where** clause packing.

See *Replication Server Administration Guide Vol 2 > Manage Warm Standby Applications*.

### *Quoted Table and Column Names and Replication Definitions*

When Replication Server packs a DML command for the target database, and if there are one or more replication definitions for the table, Replication Server decides that a table or column name should be quoted according to a replication definition if the replication definition is:

- Subscribed by a MSA database.
- Marked **send standby [ replication definition | all ] columns** for MSA databases that do not have subscriptions to a replication definition.
- Marked **send standby replication definition columns** for the standby database.

Otherwise, Replication Server packs a table or column name according to the LTL.

### *System Requirements*

RepAgent sends primary-key and quoted identifier information only with LTL version 740 or later, which is supported by Adaptive Server 15.7 and later, and Replication Server 15.7 and later.

## **Target-Scope Customized Function Strings**

In a replication system containing only Adaptive Server databases, you need not create a replication definition for a primary table or stored procedure in a warm standby environment or multisite availability (MSA) environment if the sole purpose of the replication definition is to specify a customized function string for the replicate or standby table or stored procedure.

You can create a customized function string, called a target-scope function string, directly against a replicate or standby table or stored procedure without defining a replication definition. This further reduces the replication definition requirement in a warm standby or MSA environment. Use **create function string** to create target-scope function strings and **alter function string** and **drop function string** to manage the function strings.

### *Stored Procedure and System Table Support*

Use **rs\_helpobjfstring** to display information about target-scope function strings. See *Replication Server Reference Manual > RSSD Stored Procedures > rs\_helpobjfstring*.

The `rs_targetobjs` system table stores information about target tables or stored procedures. See *Replication Server Reference Manual > Replication Server System Tables > rs\_targetobjs*.

Replication Server does not replicate the values in `rs_targetobjs` to the RSSDs of other Replication Servers. `rs_targetobjs` is in the system table services (STS) cache with

## Manage Replicated Objects Using Multisite Availability

(objname, objowner, dbid, objtype) as the STS primary cache key. Use

**sts\_full\_cache\_rs\_targetobjs** to enable or disable full caching of the table:

```
configure replication server set sts_full_cache_rs_targetobjs to {on|off}
```

The default for **sts\_full\_cache\_rs\_targetobjs** is off.

### **Warm Standby and Multisite Availability Environments**

There are several conditions that must be met before a warm standby or multisite availability (MSA) system can use target-scope function strings.

- If an MSA system subscribes to a replication definition of a table or stored procedure, Replication Server uses the function string of the replication definition, which can be a default function string or a customized function string.
- If an MSA system does not subscribe to any replication definition of a table or stored procedure, Replication Server uses the function string of a replication definition for the target object if the replication definition is marked with the **send standby all** or **send standby replication definition columns** clauses.
- A warm standby system uses the function string of a replication definition if you have marked the replication definition with the **send standby replication definition columns** clause.

If none of the conditions are met, the warm standby or MSA system uses the target-scope function string for the replicate or standby table or stored procedure, if there is one.

### **Comparison of Target-Scope and Replication Definition-Scope Function Strings**

There are several differences between target-scope and replication definition-scope function strings.

| <b>Replication Definition-Scope</b>                                                                                                                                                 | <b>Target-Scope</b>                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| A replication definition-scope function string is associated with a function-string class.                                                                                          | A target-scope function string is associated with a target (standby or replicate) database.                                                            |
| Create replication definition-scope function strings against replication definitions for function-string classes at the Replication Server that manages the replication definition. | Create target-scope function strings for target tables or stored procedures at the Replication Servers that manage the standby or replicate databases. |

| Replication Definition-Scope                                                                                                                                              | Target-Scope                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Replication Server checks whether a column or parameter for a function string is valid when the function string is created or altered against the replication definition. | Replication Server does not check the validity of a column or parameter for a function string until Replication Server converts a DML command using the function string template, and applies the command to the standby or replicate database. If the column or parameter information is incorrect, the DSI connection shuts down. You can resume the DSI connection after correcting the function string.                                                                         |
| When you create a replication definition-scope function string, Replication Server creates the full set of default function strings for the replication definition.       | When you create a target-scope function string, Replication Server creates only the function string that you specify.<br><br>The only exception is with the <b>rs_get_textptr</b> and <b>rs_writetext</b> function strings because they coexist. If there is a customized function string for <b>rs_writetext</b> , the function string for <b>rs_get_textptr</b> also exists. They both can exist as a customized function string, or as a system-created default function string. |

### Target-Scope Function String Commands

Use **create function string**, **alter function string**, and **drop function string** to create and manage target-scope function strings for replicate and standby tables and stored procedures.

Use the *[owner].table*, *stored procedure*, , and *data\_server.database* parameters to create and manage target-scope function strings.

```
{create | alter | drop} function string
 {replication_definition |
 [owner.] table |
 stored_procedure}.function[; function_string]

for {[function class]function_class |
[database] data_server.database}...
```

### Example

create the **rs\_insert** customized function string for the `dbo.authors` table at the `rdb1` target database in the `NY_DS` data server:

```
create function string dbo.authors.rs_insert
for database NY_DS.rdb1
output language
'insert authors values (
 ?au_id!new? ,
 ?au_lname!new? ,
 ?au_fname!new? ,
 ?phone!new? ,
```

## Manage Replicated Objects Using Multisite Availability

```
?address!new? ,
?city!new? ,
?state!new? ,
"00000" ,
?contract!new?)
update fn_monitor set insert_count = insert_count + 1'
```

See **create function string**, **alter function string**, and **drop function string** in *Replication Server Reference Manual > Replication Server Command* for full syntax, parameter descriptions, more examples, and usage information.

### **Listing Target-Scope Function Strings**

Use the **rs\_helpobjfstring** stored procedure to list target-scope function strings for databases such as Adaptive Server and Oracle, that support stored procedures.

To list customized function strings for replicate or standby tables or stored procedures, enter:

```
rs_helpobjfstring data_server, database, [owner.]object_name[,
function_name]
```

Suppose you create a target-scope function string for the **upd\_datetime** stored procedure:

```
create function string upd_datetime.upd_datetime
for database NY_DS.rdb1
with overwrite
output language
'update datetime set
row_num = ?row_num!param?,
datecol = ?datecol!param?,
timecol = ?timecol!param?,
ndatecol = ?ndatecol!param?,
ntimecol = ?ntimecol!param?,
comment = ?comment!param?
where
row_num = ?row_num!param?'
```

If you enter:

- `rs_helpobjfstring NY_DS,rdb1,upd_datetime`

or

- `rs_helpobjfstring NY_DS,rdb1,upd_datetime,upd_datetime`

You see:

Function String information for Target Object: 'upd\_datetime'.

| Object Name  | Object Type      | Function Name |
|--------------|------------------|---------------|
| upd_datetime | stored procedure | upd_datetime  |

| Function String Name | Output Type | Option         | System Generated |
|----------------------|-------------|----------------|------------------|
| upd_datetime         | language    | not applicable | no               |

--- Beginning of Function String Text ---

```
FString Text

update datetime set
 row_num = ?row_num!param?,
 datecol = ?datecol!param?,
 timecol = ?timecol!param?,
 ndatecol = ?ndatecol!param?,
 ntimecol = ?ntimecol!param?,
 comment = ?comment!param?
where
 row_num = ?row_num!param?

 --- End of Function String Text ---

(return status = 0)
```

See *Replication Server Reference Manual* > *RSSD Stored Procedures* > **rs\_helpobjfstring** for more examples and usage information.

## **Configuring the Replication System to Reduce Replication Definitions**

Configure the replication system to reduce replication definitions in an Adaptive Server warm standby or multisite availability environment.

See *Replication Server Reference Manual* for the full syntax of the commands and system procedures.

1. At the primary Replication Server, suspend log transfer:

```
suspend log transfer from all
```

2. At the primary data server, stop the RepAgent:

```
sp_stop_rep_agent dbname
```

3. At the primary and replicate Replication Servers, set the site version to the upgraded Replication Server version if necessary. The new site version must be 1571 or later.

```
sysadmin site_version, new site version
```

4. Drop all replication definitions that exist only to define primary keys, quoted identifiers, and customized function strings.
  - a) At the RSSD or ERSSD of the primary Replication Server, use **rs\_helpcheckrepdef** to identify such replication definitions.
  - b) At the primary Replication Server, use **drop replication definition** to drop the replication definitions listed by **rs\_helpcheckrepdef**.
5. At the primary database, create a unique index for tables that do not have a primary key or a unique index, and that do not have replication definitions.

RepAgent selects as the primary key:

- The primary index defined on a table, or
- A unique index with the lowest index identifier if there is no primary index

**Note:** If there is no primary key or unique index defined on the table, RepAgent does not send primary key information to the LTL. If the table does not have a replication definition, Replication Server treats all columns in the table, except `text`, `image`, or `unitext` columns, as primary key columns. This may degrade replication performance for the table.

- a) Execute `sp_setreptable` without any parameters to see primary key and index information for all tables that you had marked for replication with `sp_setreptable`.

| Name      | Repdef Mode | Index Mode | Primary Key         |
|-----------|-------------|------------|---------------------|
| hola      | owner_off   | no index   | hola_index          |
| inds      | owner_off   | no index   | inds_1335724833     |
| t_applied | owner_off   | no index   | t_applied_855723122 |
| t5        | owner_off   | no index   | con_pk_t5           |

You can also specify a single table. For example:

```
sp_setreptable inds
```

- b) Create a unique index for any table without a unique index or primary key.

See *Adaptive Server Enterprise > Transact-SQL Users Guide > Creating Indexes on Tables*.

- 6. If any of the replication definitions that you dropped in step 4 have customized function strings, re-create them as customized target-scope function strings at the Replication Server that controls the standby or MSA database. If there is more than one target database, re-create the function strings for each of them.
- 7. Resume log transfer and start the RepAgent.

- a) At the primary data server, start RepAgent:

```
sp_start_rep_agent dbname
```

- b) At the primary Replication Server, resume log transfer:

```
resume log transfer from all
```

**Note:** If you change the primary key for a table schema, you must suspend and resume all connections from databases that subscribe to the table and have `dynamic_sql` turned on.

## Manage Database Subscriptions

When you create a database replication definition at the primary database, you must also create a database subscription at each subscribing database.

You can use the no materialization method or the bulk materialization method. If you create a database subscription, you cannot use a **where** clause to set the criteria for subscribed data; all data is subscribed.

If you need to set criteria for particular tables or functions, you can add table or function subscriptions concurrently.

When there is a database subscription, the DSI for that connection is always treated as if for regular replication. That is, the **dsi\_replication** parameter is off, and the **dsi\_keep\_triggers** parameter is on.

When there is a database subscription—and table and function replication definitions—but there are no table or function subscriptions:

- If the table and function replication definitions contain the **send standby** clause, the database subscription honors the table or function replication definition.
- If the table and function replication definitions do *not* contain the **send standby** clause, all columns and parameters are replicated and the data is converted to the declared column and parameter datatypes.

### See also

- *Concurrently Using Database, Table, and Function Subscriptions* on page 471

## Materialization

Database subscription requires either the none method (no materialization) or the bulk method of materialization.

### Subscription Without Materialization

Use **create subscription** with the **without materialization** clause to create a database subscription when the primary and replicate databases have been synchronized prior to subscription, or activities at the primary database can be suspended.

When you use the no materialization method as described in the simple scenario for replicating a database, you can materialize the replicate databases using **bcp**, **dump** and **load**, **mount** and **unmount**, or other methods. Because Replication Server does not coordinate the initial database synchronization process, you will likely need to suspend database applications. Use this method if you are materializing the replicate database with a cross-platform **dump** and **load** (XPDL). See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Set up ASE Warm Standby Databases > Task Four: Adding the Standby Database > Determine How to Initialize the Standby Database > Cross-Platform Dump and Load* for instructions.

### See also

- *Replicating the Database in a Simple Scenario* on page 452

### Subscription with Bulk Materialization

You can use **dump** and **load** or manual coordination methods to synchronize databases.

To create a database subscription using dump and load coordination, use **define subscription** with the **use dump marker** clause. Both the primary and replicate databases and Replication Servers, must have the same server user ID, password, and role settings.

## Manage Replicated Objects Using Multisite Availability

```
define subscription sub_2
 for database replication definition repdef_1
 with primary at PDS.primary_db

with replicate at RDS.rdb
subscribe to truncate table
use dump marker
```

### Synchronizing Databases After Defining the Subscription

Synchronize databases after defining the subscription.

1. Dump *PDS.pdb*. The DSI connection to the replicate database is suspended when the dump marker reaches the replicate Replication Server. It is suspended so that no data will be replicated until you finished step 2. Replication Server activates and validates the subscription automatically when the dump marker is replicated.

---

**Warning!** Do not execute **activate subscription** or it will override the wait for dump marker at the Replication Server.

---

2. Load *PDS.primary\_pdb* to *RDS.rdb*.
3. Bring *RDS.rdb* online:  
**online database RDS.rdb**
4. Prepare *RDS.rdb* as a replicate:
  - a) In *RDS.rdb*, disable the secondary truncation point, if necessary:

```
dbcc settrunc('ltm', 'ignore')
```
  - b) Optionally, disable RepAgent:

```
sp_config_rep_agent primary_db, 'disable'
```
  - c) Dump the transaction log.
5. Resume the DSI connection to *RDS.rdb*.

## Alter Database Subscriptions

You cannot alter a database subscription directly. To alter a database subscription, delete the existing database subscription using **drop subscription**, and then create a new one.

## Drop Database Subscriptions

Use **drop subscription** to delete a database subscription.

You must include the **without purge** option so that Replication Server will not remove rows added by the subscription to the replicate. For example:

```
drop subscription sub_1a
 for database replication definition rep_1
 with primary at PDS.pdb
 with replicate at RDS.rdb

without purge ...
```



Dropping a database subscription does not affect existing table or function subscriptions. Similarly, dropping a table or function subscription does not affect existing database subscriptions. However, in this case, the replicate tables may need to be rematerialized.

See the *Replication Server Reference Manual* for complete syntax and usage information.

## View Information About Database Subscriptions

---

You can use **rs\_helpdbsub** to view information about a specific database subscription or all database replication definitions for a database or a data server.

For example, to view information about the **sub\_2B** database subscription, enter:

```
rs_helpdbsub sub_2B, dsA, db
```

See *Replication Server Reference Manual > RSSD Stored Procedures > rs\_helpdbsub* for syntax and usage information.

## Concurrently Using Database, Table, and Function Subscriptions

---

If a database subscription and table or function subscriptions coexist, the table or function subscription overrides the database subscription.

That is, Replication Server replicates the table or function according to the table or function subscription, not the database subscription.

Database subscriptions do not support the **where** clause or the **for new articles** clause. When using a database subscription, you need to create a table subscription only to:

- Use the **where** clause of a table subscription.
- Replicate a table filtered out by the database replication definition.

---

**Note:** A database subscription supports the **subscribe to truncate table** clause, but not for those tables with a table subscription.

---

- Implement autocorrection on a table.

### See also

- *Concurrently Using Database, Table, and Function Replication Definitions* on page 460

## Create and Drop Subscriptions

When database and table or function subscriptions are used concurrently, there are several procedures to follow when you create or drop those subscriptions

- If you create a database subscription that references a table with an existing table subscription, make sure you do not overwrite the replicated table when synchronizing databases:
  1. Back up the replicated table.
  2. Use **dump** and **load** to synchronize the replicate database.
  3. Copy the replicated table back into the replicate database.
- Drop a table or function subscription with suspension. After the replicate DSI is suspended, you can dematerialize or resynchronize the replicate table or function.
- When a database subscription exists that includes a table and you want to add a table subscription, make sure you define the table subscription using bulk materialization. Defining the table subscription does not stop database replication for the table. Database replication for a table stops when its table subscription is activated.
- When dropping a database subscription, you must manually purge all replicated tables that do not have table subscriptions. Replication Server does not dematerialize replicated tables.

## Replicating the Master Database in an MSA Environment

You can replicate Adaptive Server logins from one master database to another. Master database replication is limited to DDL, and the system commands used to manage logins and roles.

Master database replication does not replicate data from system tables, or replicate data or procedures from any other user tables in the master database.

Both the source Adaptive Server, and the target Adaptive Server must be the same hardware architecture type (32-bit versions and 64-bit versions are compatible) and the same operating system (different versions are also compatible).

For restrictions and limitations, and a list of supported DDL and system procedures that apply to the master database, in *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Replicated Information for Warm Standby > Use **sp\_reptostandby** to Enable Replication > Restrictions and Requirements When Using **sp\_reptostandby***, see:

- *Supported DDL Commands and System Procedures*
- *Replication of the Master Database: Limitations*

Replication Server versions 12.0 and later support master database replication with warm standby, and with MSA in Replication Server 12.6 and later. The primary or active Adaptive Server must be version 15.0 ESD #2 and later.

See *Replication Server Administration Guide Volume 2 > Manage Warm Standby Applications > Replication of the Master Database in a Warm Standby Environment for ASE* for master database replication in a warm standby environment.

1. Use **rs\_init** to set up the primary and replicate master databases.
2. Use **bcp** or manually synchronize **syslogins** and **suids** at each master database. Do not use **dump** and **load** to materialize the replicate master database.
3. Mark the primary master database:

```
sp_reptostandby master, 'all'
```

4. Stop the RepAgent on the primary master database:

```
sp_stop_rep_agent master
```

5. Configure the replication primary master database to send warm standby transactions:

```
sp_config_rep_agent master, 'send warm standby
xacts', 'true'
```

6. Restart the RepAgent on the primary master database:

```
sp_start_rep_agent master
```

7. Create a database replication definition to replicate the system procedures:

```
create database replication definition master_dbrep
with primary at PDS.master
replicate system procedures
```

8. Create a database subscription for each subscribing master database; do not materialize the data:

```
create subscription master_dbsub1
for database replication definition master_dbrep
with primary at PDS.master
with replicate at RDS.master
without materialization
```

## Replicating DDL and System Procedures

---

Learn how to use MSA to replicate DDL to nonstandby databases.

See *Replication Server Administration Guide Volume 2 > Managing Warm Standby Applications > What Information is Replicated for ASE Warm Standby Application? > Using sp\_reptostandby to Enable Replication > Supported DDL Commands and System Procedures* for a list of DDL commands supported for replication.

These constraints apply:

- When replicating system procedures, and the primary and replicate databases have different names – filter out the **sp\_config\_rep\_agent** and the **sp\_add\_user** system procedures in the database replication definition as they use database names as parameters. For example:

## Manage Replicated Objects Using Multisite Availability

```
create database replication definition myrepdef
 with primary at PDS.pdb
 not replicate system procedures in
 (sp_config_rep_agent, sp_add_user)
```

- When replicating DDL – the primary and replicate databases must have the same login names and passwords; the DSI uses the original server login name and password to log in to the replicate database.
- When replicating DDL contained in user-defined transactions – make sure that the Adaptive Server database option **ddl in tran** is set to **true**. Otherwise, the DSI will shut down when replicating DDL.
- Replication Server does not support the replication of DDL commands after **set proxy** is executed on the primary Adaptive Server. If **set proxy** is executed on the primary Adaptive Server, Replication Server returns error 5517:

```
A REQUEST transaction to database '...' failed because the
transaction owner's password is missing. This prevents the
preservation of transaction ownership.
```

---

**Note:** In a heterogeneous environment, non-Sybase data servers can replicate DDL if the Replication Agent can capture and send DDL commands in Transact-SQL or ANSI SQL (preferred).

---

To replicate DDL and system procedures:

1. Mark the primary database using **sp\_reptostandby**.
2. Set the RepAgent parameter **send warm standby xacts to true**—even if there is no standby database.
3. Create a database subscription.
4. Make sure that both the primary and replicate data servers are the same version of Adaptive Server.

### See also

- *Setting Up Bidirectional Replication Support for DDL in MSA* on page 451

## Replicate User Stored Procedures

---

To replicate user stored procedures, mark each procedure individually using **sp\_setrepproc**. Database replication definitions do not check for owner information for user stored procedures.

## Customize Function Strings

---

You can customize only those function strings that are not in the **rs\_default\_function\_class** function-string class.

For functions with replication-definition scope:

- The DSI uses **rs\_default\_function\_class** for functions that do not use a table or function replication definition with the **send standby** clause.
- Otherwise, the DSI uses the function-string class associated with the connection.

For functions with function-string class scope, the DSI always uses the function-string class associated with the connection.



# Manage Replication Schedules

You can schedule replication tasks in Replication Server. Replication Server also provides support for delaying replication by a fixed period of time.

## Schedule Replication Tasks

---

Learn to create and manage replication schedules

### Create Schedules

Use **create schedule** to create a schedule to execute shell commands at a time you specify.

For example, you can report on a specific state of the replicate database while the replicate database is frozen and not receiving data from the primary database. In this example, you can schedule replication to happen only during specific night hours, so that the processing of the next day does not change the replicate database, and reporting can occur on the data of the day before, that is frozen in the replicate database. You can do this by creating schedules to suspend and resume connections to the replicate database at specific times of the day.

Syntax:

```
create schedule sched_name as 'sched_time' [set {on | off}] for exec 'command'
```

Parameters:

- *sched\_name* – the name of the schedule you provide, which:
  - Must conform to the rules for identifiers
  - Must be unique
  - Can be 1 – 30 bytes long
- '*sched\_time*' – specifies the time and day to execute '*command*'. Provide the day and time in the restricted UNIX **cron** style with a single space separating the time and date parameters:

```
[mm] [HH] [DOM] [MON] [DOW]
```

The time and date parameters are:

| Parameter  | Description              | Values                                            |
|------------|--------------------------|---------------------------------------------------|
| <i>mm</i>  | Minutes past the hour    | 0 – 59. Use “*” to include all legal values.      |
| <i>HH</i>  | Hour in 24-hour notation | 0 – 23. Use “*” to include all legal values.      |
| <i>DOM</i> | Day of the month         | 1 – 31. Use “*” to include all days of the month. |

| Parameter  | Description       | Values                                                        |
|------------|-------------------|---------------------------------------------------------------|
| <i>MON</i> | Month of the year | 1 – 12. Use “*” to include all months of the year.            |
| <i>DOW</i> | Day of the week   | 0 – 6 with 0=Sunday. Use “*” to include all days of the week. |

- Use an asterisk “\*” to specify all valid values. For example, “17 20 \* \* \*” represents a daily schedule at 8:17 p.m.
- Use a comma “,” to separate values that are not part of a range. For example, “17 20 1,15 \* \*” represents 8:17 p.m. on the 1st and 15th of every month, where 1 and 15 are the values for the *DOM* parameter.
- Use a hyphen “-” between two values to specify a range of values, inclusive of the two values. For example, “17 20 \* \* 1-5” represents 8:17 p.m. from Monday to Friday where “1-5” are the range of values for the *DOW* parameter.
- For the *DOM*, *MON*, or *DOW* parameters, you can specify the day using both the *DOM* and *DOW* parameters. Replication Server follows all schedules you specify in the string. For example, “0 12 16 \* 1” represents 12:00 p.m. every Monday and 12:00 p.m. on the 16th of every month.
- **set {on | off}** – enables or disables the schedule when you create it. By default, the schedule is on.
- *command* – specifies the shell command, such as scripts, executables, or batch files to execute at the specified schedule.

Shell commands:

- Must be in \$SYBASE/\$SYBASE\_REP/sched for UNIX or %SYBASE%\%SYBASE\_REP%\sched for Windows
- Can include parameters delimited with a space within the shell command.
- In Windows, **create schedule** executes the command specified in the last parameter within the shell command or batch file. You must also include **stdout** to a file in the **create schedule** command line.

In shell command names, you:

- Can use only ASCII alphanumeric characters, such as A – Z, a – z, and 0 – 9
- Can use the “.”, “-”, and “\_” characters
- Cannot use the “\” and “/” characters
- Must include the .bat suffix if you are executing on Windows. For example, the name should be suspend\_conn.bat on Windows and suspend\_conn.sh on UNIX.

### Example 1

Create “schedule1” in Windows that suspends the connection to the pubs2 database in the SYDNEY\_DS data server at 12:00p.m. every Monday and 12:00p.m. on the 16th of every month:

1. Create a text file, such as sql.txt that contains the actual Replication Server command line that you want to schedule. For example, sql.txt can contain:



```
suspend connection to SYDNEY_DS.pubs2
go
```

2. Create a batch file, such as `suspend_conn.bat` in Windows that contains **isql** and relevant parameters to run the command line in `sql.txt`. For example, `suspend_conn.bat` can contain:

```
%SYBASE\OCS-15_0\bin\isql.exe -Usa -P -S SYDNEY_DS -i %SYBASE%
\sql.ini -i %SYBASE%\REP-15_5\sched\sql.txt
```

3. Create the schedule, “schedule1”:

```
create schedule schedule1 as '0 12 16 * 1' for exec
'test.bat > c:\temp\test.out'
go
```

### Example 2

Create “schedule2” to execute the `suspend_conn.sh` script in UNIX that suspends the connection to the `pubs2` database in the `SYDNEY_DS` data server every day at 8:17p.m.:

```
create schedule schedule2 as '17 20 * * *' for exec
'suspend_conn.sh'
```

### Example 3

Create “schedule3” to execute the `resume_conn.sh` script that resumes the connection to the `pubs2` database in the `SYDNEY_DS` data server every day at 7:15a.m.:

```
create schedule schedule2 as '15 7 * * *' for exec
'resume_conn.sh'
```

## System Table Support for Schedules

Replication Server provides the `rs_schedule` and `rs_scheduledtxt` system tables to store schedules you create.

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

## Alter Schedules

Use **alter schedule** to enable or disable a schedule.

Syntax:

```
alter schedule sched_name set [on|off]
```

For example, to disable `schedule1` enter:

```
alter schedule schedule1 set off
```

## Drop Schedules

Use **drop schedule** to delete a schedule from Replication Server.

For example, to delete `schedule1` enter:

```
drop schedule schedule1
```

## Display Schedules

Use **admin “schedule”** to display a schedule.

Syntax:

```
admin "schedule", ' sched_name'
```

For example, to display a schedule named schedule1, enter:

```
admin "schedule", 'schedule1'
```

---

**Note:** For **admin “schedule”**, you must enclose the **“schedule”** clause in double quotes.

---

The output is:

| Schedule Name | Schedule Time | Status | Type | Owner | Sequence | Command         |
|---------------|---------------|--------|------|-------|----------|-----------------|
| s1            | 27 * * * *    | 1      | 0    | sa    | 1        | suspend_conn.sh |

If you do not specify a schedule name, **admin “schedule”** displays all schedules in Replication Server.

## Delay Replication

You can configure Replication Server to delay replication by a fixed period of time.

You can use the replicate database as a failback system by keeping the replicate database a certain amount of time behind the primary database to provide a window of time during which to recover any human error committed on the primary database, such as a table or records dropped by mistake.

Use the **dsi\_timer** parameter with **alter connection** or **alter logical connection** to specify a delay between the time transactions commit at the primary database and the time transactions commit at the standby or replicate database. Replication Server processes transactions in the outbound queue in commit order after the period of delay is over.

You can specify a maximum delay of 24 hours. The default delay value is 00:00 which means there is no delay

After you execute **dsi\_timer** with **alter connection** or **alter logical connection**, suspend and resume the connection.

---

**Note:** Replication Server does not support time zone differences between the RepAgent at the primary database and the Replication Server with the DSI connection where you want to execute **dsi\_timer**.

---

To delay by two hours the time transactions commit at the pubs2 replicate database in the SYDNEY\_DS data server, enter:

```
alter connection to SYDNEY_DS.pubs2
set dsi_timer to '02:00'
```

## Display Delayed Replication Status

Use **admin who** to provide the status of any delay in replication that you configure with **dsi\_timer**.

**Table 36. Delayed Replication Status**

| State                       | Description                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------------------|
| Active, DSI timer           | Actively processing a command. <b>dsi_timer</b> is on.                                          |
| Awaiting Command, DSI timer | The thread is waiting for a client to send a command. <b>dsi_timer</b> is on.                   |
| Awaiting Message, DSI timer | The thread is waiting for a message from an Open Server™ message queue. <b>dsi_timer</b> is on. |



# Index

## A

activate subscription command 417, 424, 440, 443  
   for publications 440  
   publication subscription example 443

Adaptive Server  
   data compression 132  
   data compression, removing undesired data 136  
   default database 146  
   described 27  
   login name for Replication Server access 235  
   master database 146  
   master key 114  
   precomputed result sets 145  
   precomputed results sets 146  
   predicated privileges 137  
   preparing for replication 189  
   security 114  
   transfer database object ownership 147

Add Server wizard 72

adding  
   columns to a replication definition 358  
   ID server domains 90  
   primary keys 359  
   Replication Servers to existing systems 89  
   searchable columns 361, 363

admin auto\_part\_path command 54

admin verify\_repserver\_cmd command 309, 383

Advanced Security option 296

alter applied function replication definition command 382

alter auto partition path command 54

alter connection command 119, 192, 199, 287  
   changing maintenance user password 235

alter database replication definition command 458

alter function replication definition command 309, 356, 383, 393

alter replication definition command 374

alter request function replication definition command 383

alter route command 92, 107, 288, 289  
   changing passwords 234

alter user command  
   changing passwords 245

alter user command, specifying passwords with 239

altering replication definition 461

application models  
   consolidated replicate 12  
   distributed primary fragments 11  
   redistributed corporate rollup 14  
   replicated consolidated replicate 14

applied function  
   description 384

applied functions  
   prerequisites for implementing 380  
   setting up 385

article subscriptions, creating 441

articles  
   adding to publication 370  
   definition 364  
   displaying information about 369  
   dropping 371

asynchronous transaction replication 6

atomic materialization 408  
   create subscription command for 420  
   described 398, 399  
   text and image columns 431

audit\_dest parameter 299

audit\_enable parameter 299

autocorrection  
   bulk materialization 409  
   enabling for nonatomic materialization 401, 415

automatic backup, ERSSD 93

automatically resizable partitions 54  
   management and monitoring 54

## B

back-linking pointers 152

background processing 65

backup, ERSSD 100

bars  
   displaying 65  
   hiding 65

batch configuration parameter 201

batch\_begin configuration parameter 201

batching commands 191

bcp utility program 407

## Index

- bidirectional replication environment
  - creating 74
  - setting up for MSA 451
- bigdatetime and bigtime
  - mixed-version information 130
- bigdatetime and bigtime datatypes 128
- bitmap subscriptions 433
- bulk copy-in support
  - subscription materialization, changes to 411
- bulk materialization
  - autocorrection 409
  - define subscription command 423
  - described 399, 404
  - for replicated functions 392
  - methods 405
  - publication subscriptions 443
  - refreshing publication subscriptions 444
  - simulating atomic materialization 408
  - simulating nonatomic materialization 409
  - stopping primary updates 405
  - taking a snapshot 405
- buttons
  - toolbar 65
- C**
- certificate authority 296
- certificates 296
- changing
  - database connections 197
  - existing replication system 89, 91
  - ID Server login name and password 234
  - replication definitions 348
  - Replication Server login names for the RSSD RepAgent 233
  - routes 182
  - RSSD primary or maintenance user 232
  - searchable columns 361, 363
  - user passwords 245
- character sets, conversion 203
- check publication command 365, 369
- check subscription command 417, 425, 440
  - example 446
  - for articles 440
  - for publication subscriptions and articles 446
  - for publications 440
- CipherSuites 297
- class-level translations 374
  - system-defined variables 374
  - with column-level translations 377
- client application
  - described 29
- Client/Server Interfaces (C/SI), client applications
  - for 29
- column-level translations 374
  - creating 375
  - multiple replication definitions 377
- columns
  - adding to a replication definition 358
  - changing in replicated tables 360, 361, 363
  - dropping from a replication definition 358
  - identity 347
  - rs\_address datatype 433
  - specifying for replication definition 317
- command auditing 299
  - audit\_dest parameter 299
  - audit\_enable parameter 299
- command batching 191
- command\_retry command configuration parameter 201
- commands
  - alter user 239
  - configure replication server 99
  - connect 259
  - disconnect 260
  - for encrypting passwords 241
  - publications 365
  - show connection 260
  - show server 260
  - sysadmin erssd 98, 101
- Computed columns
  - deterministic expressions 344
  - nondeterministic expressions 344
  - rs\_set\_dml\_on\_computed function string 344
- concealing password input 243
- concurrency control, described 56
- configuration file 89
  - Replication Server 89
  - rs\_subcmp program 436
- configuration parameter
  - dsi\_timer 480
- configuration parameters
  - dsi\_bulk\_copy 202, 411
  - dsi\_bulk\_threshold 203
  - dynamic 96
  - ERSSD 99
  - for Replication Server 92
  - send\_standby 460
- configuration star 167

- configure
    - rs\_principal\_users.cfg 266
  - configure replication server command 92, 99, 278, 283, 289
  - Configure Replication wizard 72
  - configuring
    - environment with primary and multiple replicates 73
    - replication environment 72
    - standard warm standby environment 72
  - configuring a standard warm standby environment 72
  - connect source permission 252
  - connecting to Replication Server
    - examples 291
  - connecting to Replication Server, using network-based security 290, 292
  - connection profiles 191
    - command batching 191
  - connections 76
    - creating 76
    - defined 47
    - network-based security for 286
  - consistency
    - verifying for subscriptions 435
  - consolidated replicate application model 12
  - Contents tab 62
  - context menus 65
  - context-sensitive menus, shortcut to 65
  - controls
    - dialog box 65
  - conventions
    - style 1
    - syntax 1
  - create applied function replication definition
    - command 382
  - create article command 365, 367
  - create auto partition path command 53, 54
  - create connection command 192, 194, 286
  - create database replication definition command 457
  - create function replication definition command 383, 386
  - create function string command 412
  - create object permission 251
  - create partition command 53
  - create publication 365, 366
  - create replication definition command 309, 315, 374, 375
  - create request function replication definition
    - command 382
  - create route command 171, 288
  - create subscription command 417, 439, 441
    - and atomic materialization 421
    - direct load materialization 422
    - example for publications 441
    - for publications 367, 439
    - and nonatomic materialization 421
  - Create Subscription Without Materialization option 74, 75
  - create user command 244
    - adding Replication Server login name for RSSD RepAgent 233
    - specifying passwords with 239
  - creating
    - a replication environment object 70
    - connections 76
    - database connections 194
    - function replication definitions 386, 389
    - replicate tables 428
    - replication definitions 76, 303, 310
    - Replication Server login names 244
    - subscriptions 77, 242, 414
  - creating a bidirectional replication environment 74
  - creating partitions
    - automatically resizable 54
  - cross-platform dump and load 97
  - current\_rssd\_version configuration parameter 93
- ## D
- data availability
    - local access 6
  - data compression 132
    - removing undesired data 136
    - subscription materialization 133
    - support for Adaptive Server 132, 136
  - Data Definition Language (DDL) 449
    - replicating 473
  - data server
    - and C/SI support 15
    - ID numbers 91
    - maintenance user login names 18
    - support for heterogeneous 15, 27
    - suspending access to 198
  - database administrator, role of 20
  - database connections
    - attributes 194
    - changing attributes of 197

## Index

- creating 194
  - displaying 228
  - dropping 227
  - information for 194
  - information required 194
  - managing 189, 224
  - monitoring 228
  - suspending 198
- database logs
- truncation 57
- database replication definitions 457
- database replication filters 459
- database schema, replication definitions 349
- Database Subscription Resolution Engine (DSRE) 459
- database subscriptions
- altering 470
  - dropping 470, 472
  - managing 468
  - with table and function subscriptions 471
- databases
- managed by Replication Server 229
  - preparing for replication 189
- datatype classes
- rs\_db2\_dt\_class 375
  - rs\_hanadb\_dt\_class 375
  - rs\_iq\_dt\_class 375
  - rs\_mssql\_dt\_class 375
  - rs\_oracle\_dt\_class 375
  - rs\_sqlserver\_dt\_class 375
  - rs\_udb\_dt\_class 375
- datatype definitions 376
- for Microsoft SQL Server datatypes 377
- datatypes
- identity columns 347
  - rawobject and rawobject in row 333
  - rs\_address 346, 433
  - timestamp columns 347
- db\_packet\_size configuration parameter 201
- DB2 databases, function-string class 16
- DDL replication
- granular permissions, impact of 143
- declared datatypes 375
- dedicated routes 170
- default database 146
- deferred name resolution 131
- deferred\_name\_resolution configuration parameter 202
- define subscription command 417, 440, 443
- and bulk materialization 423
  - creating publication subscriptions 443
  - for publications 440
  - publications subscription example 443
  - using with replicated functions 387
- Defined Subscription for Bulk Materialization
- option 74, 75
- defragmenting ERSSD 101
- delaying replication 480
- status 481
- deleting an object 69
- design considerations, replication system 304
- Details list 64
- direct load materialization
- described 399
- direct routes 167
- direct\_load option 422
- directory services 36
- disable
- replication system 46
  - user authentication 46
- disabling 479
- disallowed\_prev\_passwords configuration parameter 247
- disconnecting
- from a replication environment 71, 72
- disk\_affinity configuration parameter 173, 202
- displaying
- bars 65
  - database connections 228
  - databases with subscriptions 435
  - DSI thread status 229
  - icons 65
  - replication definitions 349
  - RSI thread status 186
  - subscription information 435
  - users of replication system 257
  - users' permissions 258
- displays
- updating 67
- distributed data models
- corporate rollup 11
  - custom design 7
  - distributed primary fragments 11
  - redistributed corporate rollup 11
- distributed database system and Replication Server 7



- distributed primary fragments, consolidated
  - replicate application model 13
- drop article command 366, 371
- drop auto partition path command 54
- drop connection command 107, 227
- drop database replication definition command 459
- drop function replication definition 394
- drop function replication definition command 382
- drop publication command 365, 370, 371
- drop replication definition command 309, 362
- drop route command 107, 108
- drop schedule command 479
- drop subscription command 106, 417, 426, 440, 445
  - example 445
  - example of 445
  - for articles 440
  - for publications 440
  - function replication definitions 427
  - table replication definitions 427
- drop user, dropping login names 250
- drop\_repdef clause 371, 372
- dropping
  - a replication environment object 71
  - columns from a replication definition 358
  - database connections 227
  - databases from the ID Server 228
  - function replication definitions 394
  - primary keys 359
  - replication definitions 362
  - Replication Server login names 250
  - Replication Servers from existing system 106, 109
  - routes 184
  - searchable columns from the searchable columns list 359, 363
  - subscriptions 242
- dropping routes, restrictions 184
- DSI threads
  - described 52
  - displaying 229
  - scheduler 52
- dsi\_alt\_writetext configuration parameter 202
- dsi\_bulk\_copy connection parameter 202, 411
- dsi\_bulk\_threshold connection parameter 203
  - See also bulk copy-in support
- dsi\_charset\_convert configuration parameter 203
- dsi\_check\_lock\_wait configuration parameter 203
- dsi\_cmd\_batch\_size configuration parameter 204
- dsi\_cmd\_prefetch configuration parameter 204
- dsi\_cmd\_separator configuration parameter 204
- dsi\_command\_convert configuration parameter 205
- dsi\_commit\_check\_locks\_intrvl configuration parameter 205
- dsi\_commit\_check\_locks\_max configuration parameter 206
- dsi\_commit\_control configuration parameter 206
- dsi\_compile\_enable configuration parameter 207
- dsi\_compile\_max\_cmds configuration parameter 208
- dsi\_dataserver\_make configuration parameter 208
- dsi\_exec\_request\_sproc configuration parameter 208
- dsi\_fadeout\_time configuration parameter 209
- dsi\_ignore\_underscore configuration parameter 209
- dsi\_isolation\_level configuration parameter 209
- dsi\_keep\_triggers configuration parameter 209
- dsi\_large\_xact\_size configuration parameter 210
- dsi\_max\_cmds\_in\_batch configuration parameter 210
- dsi\_max\_cmds\_to\_log configuration parameter 210
- dsi\_max\_text\_to\_log configuration parameter 210
- dsi\_non\_blocking\_commit configuration parameter 211
- dsi\_num\_large\_xact\_threads configuration parameter 211
- dsi\_num\_threads configuration parameter 211
- dsi\_partitioning\_rule configuration parameter 211
- dsi\_proc\_as\_rpc configuration parameter 212
- dsi\_quoted\_identifier configuration parameter 213
- dsi\_replication configuration parameter 214
- dsi\_replication\_ddl configuration parameter 143
- dsi\_row\_count\_validation parameter 214
- dsi\_serialization\_method configuration parameter 215
- dsi\_sqz\_max\_cache\_size configuration parameter 216
- dsi\_stage\_all\_ops configuration parameter 216
- dsi\_text\_convert\_multiplier configuration parameters 217
- dsi\_text\_max\_xacts\_in\_group configuration parameter 210
- dsi\_timer configuration parameter 217, 480
- dsi\_top1\_enable configuration parameter 218, 311
- dsi\_xact\_group\_size configuration parameter 219

## Index

- dump and load coordination 74, 75
- dump command 407
- dump\_load configuration parameter 219
- dynamic\_sql configuration parameter 219
- dynamic\_sql\_cache\_management configuration parameter 220
- dynamic\_sql\_cache\_size configuration parameter 220

## E

- enabling RepAgent 112
- encrypted columns, replicating 344
- encrypted data, replicating 345
- encrypted passwords
  - sending 240
- enhancements
  - for Replication Server performance 52
  - Replication Server gateway 259
- environment
  - three-tier 79
  - two-tier 70
- error classes
  - Open Server gateway 16
- errsd\_backup\_dir 99
- errsd\_backup\_interval 99
- errsd\_backup\_start\_time 99
- errsd\_ra 99
- ERSSD
  - automatic backup 93
  - created by rs\_init program 90
  - media failure, recovery from 103
  - recovery instructions 102
  - recovery procedures 103
  - routing 100
  - use isql to execute 86
- ERSSD (Embedded Replication Server System Database) 98
- ERSSD (Embedded Replication Server System Database), configuration file 98
- ERSSD backup directory path 98
- ERSSD configuration parameters 99
- ERSSD database file path 98
- ERSSD transaction log file path 98
- ERSSD transaction log mirror file path 98
- errsd\_backup\_start\_date 99
- ERSSD, backup directory files 100
- ERSSD, configuration parameters in rs\_init table 92
- ERSSD, files, moving 101

- ERSSD, users 101
- Event Log pane 65
- event triggers 80
  - creating 80
- examples
  - assigning domain ID numbers 91
  - atomic materialization 400
  - routing 174
  - rs\_subcmp configuration file 436
- exceptions log
  - transactions written to 57
- exec\_cmds\_per\_timeslice configuration parameter 219
- exec\_nrm\_request\_limit configuration parameter 220
- exec\_sqm\_write\_request\_limit configuration parameter 220
- execute as clause
  - impact on stored procedure replication 143
- executing
  - RCL commands 85
  - scripts with isql 87
- ExpressConnect for HANA DB 29
- extended limits 126, 325, 326, 329
  - more columns 326
  - wide columns 326
  - wide data 326
  - wide messages 326

## F

- Failover support 97
- fault tolerance, achieving 6
- features
  - background processing 65
  - Details list 64
  - Event Log pane 65
  - script editors 66
- files
  - interfaces 36
  - moving, ERSSD 101
  - Replication Server configuration 89
  - Replication Server run file 88
- folder icons 63
- for new articles clause 442
- function replication definition 33
  - modifying 393
- function replication definitions
  - altering 392
  - commands for managing 382

- dropping 394
- subscribing to 392
- function strings
  - changing replication definitions 357
  - customized, reducing replication definitions
    - for 463
  - customizing 475
  - defined 48
  - for Java columns 334
  - target-scope functions strings, reducing
    - replication definitions for 463
  - variables 48
- Function strings
  - rs\_set\_dml\_on\_computed 344
- function-string classes
  - defined 49
  - for Adaptive Server databases 16
  - for DB2 databases 16
  - open architecture 16

## G

- grant command 191, 251, 256
- granting manage database permission to dbo 139
- granting permissions
  - for maintenance user 139
- granular permissions
  - creating maintenance user 141
  - creating maintenance user role 139
  - DDL replication, impact on 143
  - granting permissions to maintenance user 139
  - manually configuring maintenance user
    - permissions 138
  - permissions required for maintenance user
    - 140
  - Replication Server support for 138
  - rs\_init error message 142
  - rs\_maint\_user\_role 139
  - user database upgrading, impact on 142
- guidelines
  - for creating multiple ID Server domains 91

## H

- ha\_failover configuration parameter 95, 97
- HDS. See heterogeneous datatype support 372
- help
  - topic 62
- help contents 62

- heterogeneous data servers 474
- heterogeneous datatype support
  - class-level translations 374
  - column-level translations 374
  - data servers 372
  - overview 373
  - procedure for 374
- hiding
  - bars 65
  - icons 65

## I

- icons
  - displaying 65
  - folder 63
  - hiding 65
  - object 64
- ID numbers
  - data servers 91
  - Replication Server 91
- ID Server 89
  - adding server domains 90
  - assigning ID numbers 91
  - dropping a database from 228
  - guidelines 91
  - login name 25
  - login name and password 234
  - network-based security for 289
  - requirements 25
  - specifying domain ID numbers 91
- id\_msg\_confidentiality parameter 289
- id\_security\_mechanism parameter 289
- id\_server configuration parameter 93
- identity columns 347
- image datatype
  - changing replication for 341
  - overview of replication 336
- IMDB 153
- in-memory databases 153
- inbound queue
  - defined 50
- inconsistencies
  - correcting 436
  - locating 436
  - occurring in tables 436
  - resulting from skip transaction clause 224
- incremental data transfer 136
- indirect routes 167

## Index

initial\_password\_expiration configuration  
parameter 248

interfaces file 36  
defined 36  
requirements 36

isql interactive SQL utility 86, 240  
executing RCL commands 85  
executing scripts 87  
to execute ERSSD 86

isql program 89

## J

java columns, replicating 332  
Java datatypes 333

## K

Kerberos  
considerations 40  
network 37  
protocol 37

keyboard shortcuts 64  
keytab file 279, 290

## L

large identifiers 127

LDAP  
authentication 41  
lookup 41  
password 42  
server 41  
user 41  
user authentication 42

LDAP server 40  
Open Client/Server 41

libtcl.cfg file 263

limitations  
maintenance user security configuration 237

list  
Details 64

LOB datatypes  
limitations 343  
partial update 343

log  
event 65

log transfer  
resuming 119

suspending 117, 119

Log Transfer Language (LTL) 28

log truncation 58

log truncation, Adaptive Server 57

logical connections 73

login names

applied functions 243

applied stored procedures 243

creating for maintenance user 192

creating for Replication Server 244

data server 18

dependencies 231, 243

displaying maintenance user 192

dropping Replication Server 250

ID Server 25, 234

list of commands for managing 243

passwords

requirements for Replication Server 244

Replication Server 18

Replication Server for RSSD RepAgent use  
233

for Replication Server use 234

request functions 242

request stored procedures 242

RSSD maintenance user 232

RSSD primary user 232

for subscriptions 242, 415

login scripts

precomputed result sets, enabling 146

## M

maintenance user 73–75

changing passwords 235

creating in user database 141

described 235

displaying list of 192

granting database access 192

granting required granular permissions 139

granting roles 141

granular permissions required 140

login names 18, 192

password expiration 235

password expiration, setting 237

password hiding, disabling 238

password hiding, enabling 236

password status, displaying 236

password, hidden 235

recovery after RSSD failure 238

required permissions 191, 192

- rs\_init error message 142
  - RSSD 232
  - security configuration limitations 237
  - manage database replication definition command 457
  - managing
    - RepAgent 111
    - replicated tables 303
    - stable queues 51
    - supporting Adaptive Server 111
  - mapping security-system login 295
  - marking data for replication 456
  - master database 146
  - replication 472
  - master database replication
    - granular permissions, impact of 143
  - master key for Adaptive Server 114
  - materialization 74, 75
  - materialization method 73
  - materialization methods
    - for function replication definitions 442
  - materilazation
    - methods 398
  - max\_failed\_logins configuration parameter 249
  - max\_password\_len configuration parameter 246
  - md\_sqm\_write\_request\_limit configuration parameter 221
  - media failure, ERSSD, recovery 103
  - memory\_max configuration parameter 223
  - menus
    - context 65
  - migrate RSSD 97
  - migration
    - replication system 44
    - user authentication 44
  - min\_password\_len configuration parameter 246
  - minimal columns
    - specifying for replication 311, 321
  - minimal DML logging in IMDB 161
  - minimum\_rssd\_version configuration parameter 93
  - mixed version
    - multisite availability 450
  - mixed versions
    - replication system 17
  - mixed-version systems
    - Adaptive Server 17
    - restrictions 17
  - mnemonics 64
  - monitoring
    - database connections 228
    - dropped routes 184
    - routes 186
  - monitoring of status 66
  - monitoring, Replication 30
  - more columns 326
  - mount command 405
  - move primary command 106
    - routing requirements 166
  - moving ERSSD files 101
  - msg\_confidentiality parameter 282
  - msg\_integrity configuration parameter 282
  - msg\_origin\_check configuration parameter 282
  - msg\_replay\_detection configuration parameter 282
  - msg\_sequence\_check configuration parameter 282
  - multiple ID Server domains, guidelines for creating 91
  - multiple replication definitions per table 327
  - MultiSite Availability
    - reducing replication definitions 464
  - multisite availability (MSA) 32, 397
    - advantages of 449
    - bidirectional replication environment 450, 451
    - bulk materialization 452
    - concurrent replication definitions 460
    - data replication 456
    - database replication definitions 457
    - dropping database replication definitions 459
    - function strings 475
    - heterogeneous data servers 474
    - marking data 456
    - master database replication 472
    - mixed version 450
    - replicating tables and functions separately 453
    - replicating the database 452
    - resynchronizing tables 458
    - set up 449, 452–454
    - warm standby 454
  - mutual\_auth 282
- ## N
- name space, replication definition 316
  - naming replicated tables 363
  - network security
    - maintaining 292

## Index

- network-based
  - security features 19
- network-based security
  - activating 278
  - altering security mechanism 293
  - configuring services 279
  - credential 261
  - disabling 293
  - environment variables for 265
  - global settings 284
  - how it works 261
  - logging in 290
  - mapping login 295
  - message protection 262
  - parameters 282
  - pathways 280
  - planning for 283
  - potential security breach 296
  - requirements 262
  - requirements and restrictions 262
  - restrictions 262
  - setting up 263
  - using multiple security mechanisms 295
  - viewing information about 294
- networks
  - maintain security 292
- new features
  - background processing 65
  - Details list 64
  - dynamic configuration 96
  - Event Log pane 65
  - manual data reconciliation 437
  - rs\_subcmp performance enhancement 437
  - schema comparison 437
  - script editors 66
- no materialization method
  - described 399
  - describing 403
  - requirements for using 403
- non-ASE servers 190
- non-ASE support
  - command batching 191
  - connection profiles 191
- nonatomic materialization 415
  - autocorrection 415
  - described 398, 401
  - text and image columns 431
- num\_threads configuration parameter 224

## O

- object icons 64
- object properties 68
- object property sheets 68
- object tree 63
  - moving through 64
- objectid.dat file 263, 264
- objects
  - creating 70
  - creating in Sybase Central 68
  - deleting 69
  - dropping 71
  - selecting 64
- online help 62
- Open Client Client-Library 85
- Open Server gateway
  - creating for Replication Server 15
- oserver configuration parameter 93
- outbound queue, defined 50
- owner status of table
  - checking 331
  - modifying 331

## P

- parameters
  - RepAgent configuration 90
- parameters, stored procedure
  - adding to replicated functions 393
- partial update
  - LOB datatypes 343
- partitions
  - automatically growing and shrinking 54
  - guidelines for choosing 53
- password
  - forgotten sa password 250
  - lost sa password 250
- password administration in rs\_init 239
- password encryption
  - extended support 241
  - maintenance user 235
  - replication system 18
- password expiration
  - maintenance user 235, 237
- password hiding
  - maintenance user 236
- password protection, disabling
  - maintenance user 238

- password security administration
  - password parameters, setting 246
  - password resetting, sa 250
- password\_expiration configuration parameter 248
- password\_lock\_interval configuration parameter 249
- password\_lowercase\_required configuration parameter 246
- password\_numeric\_required configuration parameter 246
- password\_special\_required configuration parameter 247
- password\_uppercase\_required configuration parameter 246
- passwords
  - alter user command 245
  - applied functions 243
  - applied stored procedures 243
  - changing 245
  - changing for maintenance user 235
  - changing for Replication Server in RSSD RepAgent 233
  - changing for RSSD primary user 233
  - concealing input 243
  - dependencies 235, 243
  - for RepAgent use 233
  - ID Server 234
  - for Replication Server use 234
  - request functions 242
  - request stored procedures 242
  - specifying with alter user command 239
  - specifying with create user command 239
  - subscriptions 242
- performance
  - replicating local data 6
  - routing 169
- performance improvements for Adaptive Server replication 152
- permissions
  - creating subscriptions 414
  - displaying for users 258
  - dropping subscriptions 426
  - for adding Replication Server and Adaptive Server Enterprise 70
  - granting 256
  - granting database access for maintenance user 192
  - maintenance user 192
  - managing for Replication Server 251, 258
  - revoking 257
  - sa, sa\_role, sso\_role 70
  - subscription requirements 415
  - summary of commands for 253
  - system for 18
- precomputed result sets
  - support for Adaptive Server 145
- precomputed results sets
  - enabling replication support for earlier Replication Server versions 146
- predicated privileges
  - support for Adaptive Server 137
- preparing
  - three-tier 79
- preparing for replication
  - Adaptive Server databases 189
- prev\_min\_rssd\_version configuration parameter 93
- prev\_rssd\_version configuration parameter 93
- primary data 57
  - failure to update 57
- primary data server
  - subscription requirements 415
- primary databases
  - required permissions 193
  - subscription requirements 415
- primary key
  - adding or dropping 359
  - defined 31, 311
  - requirement for unique 305
- primary key clause 320
- primary key columns
  - restrictions on updating 305
- Primary LDAP URL
  - authentication 45
  - SSL 45
- primary subscribe permission 251
- primary tables
  - subscription requirements 414
- primary user
  - RSSD 232
- principal user 278
- processing
  - background 65
- properties
  - object 68
- property sheets 68
- publication subscriptions 438
  - activating 443
  - bulk materialization method 443

## Index

- creating 441
  - defining 443
  - definition 364
  - dropping 445
  - monitoring 445
  - refreshing 444
  - restrictions 439
  - specifying materialization methods 442
  - status information 446
  - validating 444
  - publications 372
    - adding articles 370
    - altering 370
    - commands 365
    - creating at the command line 366
    - definition 364
    - displaying information about 369
    - dropping 370
    - dropping replication definitions 371
    - for stored procedures 394
    - from Sybase Central 364
    - procedure for creating 364
    - RCL commands for 365
    - viewing information about 368
  - published datatype 375
- ## Q
- queue data 77
    - viewing 77
  - queue data dialog box 78
  - quiescing
    - procedure for Replication Server 105
    - replication system 105
  - quoted identifiers
    - always add quotes 314
    - dsi\_quoted\_identifier system function 313
    - enabling 313
    - marking 313
    - mixed-version restrictions 315
    - replication definitions 312
    - without replication definition support 314
- ## R
- RCL commands 118
    - activate subscription command 417
    - admin verify\_repserver\_cmd command 309, 383
    - alter applied function replication definition command 382
    - alter connection command 192, 199
    - alter function replication definition command 383
    - alter function string command 394
    - alter replication definition command 309, 356
    - alter request function replication definition command 383
    - check subscription command 417, 425
    - create applied function replication definition command 382
    - create connection command 192, 194
    - create function replication definition command 383, 386, 389
    - create replication definition command 309, 315
    - create request function replication definition command 382
    - create route command 171
    - create subscription command 417, 421
    - define subscription command 417, 423
    - drop connection command 227
    - drop function replication definition command 382, 394
    - drop replication definition command 309
    - drop route command 184
    - drop subscription command 417, 426
    - drop user command 250
    - executing command 85
    - grant command 256
    - resume connection command 224
    - resume route command 177
    - revoke command 257
    - set autocorrection command 415
    - shutdown command 89
    - suspend connection command 198
    - suspend log transfer command 117
    - suspend route 176
    - sysadmin dropdb command 228
    - sysadmin purge\_route\_at\_replicate command 185
    - table of permissions 253, 256
    - validate subscription command 417
  - recovery instructions, ERSSD 102
  - recovery procedures, ERSSD 103
  - recovery, from media failure, ERSSD 103
  - redistributed corporate rollup application model 14
  - reducing ERSSD size 101



- reducing replication definitions
  - warm standby 462, 463
- relaxed-durability databases 153
- religation definitions
  - replicate minimal columns clause 321
- removing undesired data 136
- Rep Agent user 73
- rep\_as\_standby configuration parameter 221
- RepAgent 111
  - configuration parameters 113
  - described 28
  - disabling 116
  - enabling for databases 112
  - enabling on Adaptive Server 112
  - error messages 116
  - extended limits 126
  - network security for 117
  - parameters 90
  - parameters, viewing information on 120
  - for RSSD 233
  - secondary truncation point 57
  - setting up 111
  - starting 114
  - starting up 115
  - status and configuration information 119
  - stopping 115
  - suspending 117
  - thread user status 120
  - truncation point 58
  - viewing information on 119
- RepAgent Executor 219
- RepAgent user 74, 75
- replicate databases
  - upgrading to primary databases 224
- replicate Replication Server
  - subscription requirements 415
- replicate tables
  - requirements for subscriptions 414
- replicate\_minimal\_columns configuration
  - parameter
    - for database connection 221
- replicated consolidated replicate application model 14
- replicated function
  - creating 394
  - modifying 394
- replicated functions
  - adding parameters 393
  - adding searchable parameters 393
- benefits 33
  - described 34, 384
  - dropping 394
  - modifying 392
  - restrictions 380
  - subscribing to 386
- replicated stored procedures
  - enabling for replication 391
  - login and password dependencies 242
  - replicated function delivery 384
- replicated tables
  - changing 362
  - changing searchable columns 361, 363
  - commands for modification 309
  - enabling for replication 329
  - failed updates 57
  - renaming primary and replicate copies 363
  - requirements 31
  - subscribing to 397
- replicated tables, introduction to 303
- replicating
  - data, large batch of 52
  - java columns 332
  - large objects to non-ASE servers 337
  - LOB datatypes 342
  - master database in an MSA 472
  - partitioned tables 419
  - timestamp columns 347
- replicating encrypted columns 344
- replicating encrypted data 345
- replicating tables overview 35
- replication
  - ASE shared-disk cluster 130
  - delaying 477
  - scheduling 477
  - scheduling tasks 477
  - specifying a delay 480
- Replication Agent
  - described 28
  - open architecture 15
  - requirements 27
- replication definition
  - altering 461
  - coordinating with transfer database object
    - ownership 149
- replication definitions 76
  - altering 349
  - change request process 350
  - changing 348

## Index

- commands for managing 309
- creating 310
- datatypes 317
- defined 32
- described 310
- dropping 362
- dropping from articles 371
- dropping from publications 371
- error handling 353
- examples 315
- executing directly at the primary database 351
- extended limits 325
- functions 384
- name space 316
- for distributed primary fragments 11, 12
- primary key 320
- quoted identifiers 312
- requirements for creating subscriptions 414
- rs\_address datatype 433
- searchable columns 320
- skipping faulty commands 353
- suspending DSI 352, 393
- text and image columns 324
- text or image columns 336
- use case for change request process 354
- using 310, 362
- verifying replication definition RCL
  - commands 353
- warm standby 462
- replication definitions, reducing
  - configuring 467
  - target scope functions strings, using 463
- replication environment
  - configuring 72
  - configuring with primary and multiple replicates 73
  - disconnecting from 71, 72
  - setting up 69
- replication environment object
  - creating 70
  - dropping 71
- Replication Server 24
  - about 5
  - adding to an existing system 89
  - advantages of replicating local data 6
  - configuration file 89, 285
  - configuring rs\_config system table 92
  - connections 36
  - described 24
  - distributed data models 7
  - dropping from existing system 106, 109
  - executable program 88
  - general description 5
    - and heterogeneous data servers 15
  - ID numbers 91
  - introduction 5
  - list of databases managed by 229
  - login name for Adaptive Server use 235
  - login name for RSSD use 235
  - managing 83, 105
  - managing login names 243
  - managing objects 75
  - objects 75
  - permissions 251, 257
  - primary copy model 8
  - principal name 37
  - quiescing 105
  - role in a distributed database system 7
  - run file for 88
  - security 258
  - shutting down 89
  - starting 88
  - stopping 89
  - subscription requirements 415
  - system data flow 9
  - technical overview 23
  - transaction handling 49
  - transfer database object ownership 147
- Replication Server gateway
  - cascading connection 259
  - connection dropping 260
  - connection tracking 260
  - enable, to 259
  - limitation 260
  - product version requirement 260
- Replication Server programs
  - repserver 88
  - rs\_init 89
  - rs\_subcmp 436
- Replication Server support
  - conditions and limitations for transfer database object ownership 150
- Replication Server System Database (RSSD)
  - described 26
  - login names 232
  - maintaining 92
  - managing 96
  - RepAgent for 233

- requirements 27
- rs\_helpdb stored procedure 229
- rs\_maintusers 258
- rs\_users 258
- system tables 26
- users 232
- replication system
  - components 23
  - domains 90
  - guidelines for creating multiple domains 91
  - open architecture 15
  - planning 304
  - preparing 306
  - quiescing 105
  - Replication Server 24
  - security 231
  - setting up 83
  - tasks and roles 20
- replication system administrator, role of 19
- replication\_role
  - granting to maintenance user 141
- repsrv command 88
- request function
  - description 387
- request functions
  - defined 34
  - login names and passwords 243
  - permissions needed at primary 193
  - prerequisites for implementing 380
  - setting up 388
- request stored procedures
  - login names and passwords 243
  - primary copy model 8
- restrictions
  - on replicated data 305
- resume connection command 224, 287, 409, 411
- resume log transfer command 118
- resume route command 177, 289
- resuming
  - log transfer 118
  - RepAgent 118
  - routes 176
- resynchronizing replicate tables 458
- retrieving data from primary database
  - using ASE dump and load, select, or bcp commands 407
  - using ASE mount command 405
- revoke command 251, 257
- RMS
  - adding servers 80
  - connecting to 79
  - monitoring replication environment 79
  - three-tier management solution 69
  - viewing objects 80
- RMS (Replication Monitoring Services) 31
- roles and responsibilities, for Replication Server 19
- roll-up
  - consolidated replicate application model 12
  - consolidated replicate as primary application model 14
- route
  - topology, changing 178
- route version
  - between Replication Servers 185
- routes
  - changing 177, 182
  - create 170
  - creating login names 172
  - creating, rules for 170
  - dedicated 170
  - defined 47
  - determining 166
  - direct 167
  - dropping 184
  - indirect 167
  - managing 165, 183
  - monitoring creation of 186
  - network-based security for 288
  - purging 108
  - resuming 176
  - rules 166
  - subscriptions 169
  - suspending 176
  - unsupported 170
  - upgrading 185
- routing
  - ERSSD 100
  - examples 174
  - overlapping subscriptions 169
  - preparation 165
  - prerequisites 165, 166
  - rules 166
  - schemes 167
- row migration
  - text and image columns 431
- rs password configuration parameter 114

## Index

- rs\_address datatype 346
  - rs\_config system table 94
  - rs\_databases system table 229
  - rs\_ddlsession\_resetting system function 145
  - rs\_ddlsession\_setting system function 145
  - rs\_helpdb stored procedure 106, 229
  - rs\_helpobjfstring system procedure 466
  - rs\_helppub stored procedure 366, 369, 446
  - rs\_helpprepdb stored procedure 435
  - rs\_helproute stored procedure 186
  - rs\_helproute.using to monitor dropped routes 184
  - rs\_helpuser stored procedure 257
  - rs\_idnames system table
    - dropping database from 228
  - rs\_init
    - creates ERSSD 90
    - error message, when adding a database 142
    - granular permissions, impact of 138, 142
    - maintenance user permissions, user database and RSSD 138
    - password administration 239
  - rs\_init program 92
  - rs\_lastcommit system table 224
    - permissions 192
  - rs\_maint\_user\_role
    - creating in master database 139
    - granting permissions to 139
    - granting set session authorization permission 139
    - granting to maintenance user 141
  - rs\_maintusers system table 258
  - rs\_marker stored procedure 193
  - rs\_schedule system table 479
  - rs\_scheduledtxt system table 479
  - rs\_send\_repserver\_cmd stored procedure 309, 383
  - rs\_set\_dml\_on\_computed function string 344
  - rs\_setproxy function string 292
  - rs\_subcmp program 436
  - rs\_subscriptions system table 443
  - rs\_update\_lastcommit stored procedure 193
  - rs\_users system table 258
  - RSI threads
    - described 52
    - displaying 186
  - rsi\_batch\_size configuration parameter 173
  - rsi\_fadeout\_time configuration parameter 173
  - rsi\_packet\_size configuration parameter 173
  - rsi\_sync\_interval configuration parameter 173
  - rsi\_xact\_with\_large\_msg configuration parameter 173
  - RSSD 70
    - network-based security for 285
  - RSSD failure
    - maintenance user resetting 238
    - recovery 238
  - RSSD stored procedures
    - rs\_send\_repserver\_cmd stored procedure 309, 383
  - rssd\_error\_class configuration parameter 93
  - run file, Replication Server 88
- ## S
- sa permission 70, 251, 253
  - sa\_role permission 70
  - save\_interval configuration parameter
    - for database connection 222
    - for route 173
  - scenario
    - Adaptive Server 266
    - authentication 266
    - Kerberos 266
    - Replication Server 266
  - schedules 479
    - admin"schedule" command 480
    - alter schedule command 479
    - altering 479
    - create schedule command 477
    - creating 477
    - deleting 479
    - displaying 480
    - dropping 479
    - enabling 479
    - system tables support 479
  - scheduling replication tasks 477
  - script editors 66
  - scripts
    - executing in isql 87
    - replication definition examples 428
  - searchable columns 320
    - adding 361, 363
    - dropping from the searchable columns list 359, 363
    - restrictions 321
  - searchable columns clause 311
  - searchable parameters
    - adding to replicated functions 393

- secondary truncation point
  - and disabling RepAgent 116
  - described 58, 59
- secure sockets layer 296
- security
  - master key for Adaptive Server 114
  - network-based 261
  - password parameters 246
  - password resetting, sa 250
  - recommendations 301
  - RepAgent 117
  - Replication Server 231, 258
  - replication system 18
  - rs password configuration parameter 114
- security features
  - network-based 19
  - of Replication Server 18
  - third-party 19
- security mechanism, altering 293
- security mechanisms
  - CyberSafe Kerberos 262
  - DCE 264
  - Transarc DCE 262
- security services, configuring 279
- security\_mechanism 282
- security, network-based 243
- select command 407
- select with holdlock clause 442
- send\_emc\_pw configuration parameter 94
- send\_enc\_password configuration parameter 240
- send\_standby configuration parameter 460
- service key for Adaptive Server 114
- set autocorrection command 415
- set proxy command 277, 292
- set security\_mechanism parameter 293
- set session authorization permission, granting to
  - rs\_maint\_user\_role 139
- setting up network-based security 263
- Setting up RepAgent 111
- shortcut menus 65
- shrinking databases
  - Replication Server support for 152
- shutdown RCL command 89
- shutting down
  - Replication Server 89
- simple\_passwords\_allowed configuration
  - parameter 247
- simulating 408
- simulating atomic materialization 408
- skip transaction clause 224
- sp\_dropalias Adaptive Server system procedure
  - 143
- sp\_role system procedure 191
- sp\_setreplicate system procedure
  - marking rs\_marker for replication 226
- sp\_setrepproc system procedure 385
  - marking stored procedures for replication 391
  - using for applied function 385
  - using for request function 389
- sp\_stop\_rep\_agent command 107, 115
- special datatypes 346
- Specifying
  - principal name 38
  - sybase\_rs\_principal 38
- SSL 296
  - certificate authority 296
  - enabling on Replication Server 298
  - handshake 297
  - managing 296
  - on Replication Server 297
  - overview 296
  - requirements for 297
  - setting up 298
  - trusted roots file 297
- sso\_role permission 70
- stable queues
  - atomic materialization 400
  - described 50
  - disk files 54
  - management 51
  - requirements 53
  - for routes 170
- stage\_operations configuration parameter 222
- star configuration
  - described 167
- starting
  - Replication Server 88
  - Sybase Central 61
- static parameters
  - configuring 95
  - modifying with configure replication server
    - command 95
- status
  - monitoring 66
- status bar 65
- stored procedure replication
  - impact of execute as clause 143

## Index

- stored procedures
  - marking for replication using `sp_setrepproc` 391
  - publications 395
  - `rs_helpdb` 229
  - `rs_helprep` 349
  - `rs_helrepdb` 435
  - `rs_helreptable` 349
  - `rs_helrepversion` 349
  - `rs_helproute` 186
  - `rs_helpsub` 435
  - `rs_helpuser` 257
  - `rs_update_lastcommit` 193
- `sub_sqm_write_request_limit` configuration parameter 222
- subscribe to truncate table clause 441
- subscribing
  - to replicated tables 397
  - to function replication definitions 392
- subscription dematerialization
  - methods 426
  - phases 413
  - processing 411
  - with purge 412
- subscription materialization
  - defined 32
  - methods 415
  - phases 413
  - text and image columns 431
- subscription materialization queue
  - defined 50
- subscription migration
  - `rs_address` columns 434
- subscription resolution engine (SRE) 459
- subscriptions 77
  - adding to a publication subscriptions 442
  - bitmap 433
  - commands for managing 416
  - defined 397
  - displaying 435
  - dropping 242, 426, 472
  - login names and password dependencies 242
  - overlapping 169
  - permissions for creating 252
  - preparations for creating 414
  - primary fragments 11, 12
  - removing rows manually 427
  - requirements 414
  - user permission requirements 253
  - verifying consistency 435
- support
  - bidirectional replication 451
  - bigdatetime and bigtime datatypes 128
  - deferred name resolution 131
  - extended password encryption 241
  - for ASE shared-disk cluster 130
  - IMDB 153
  - IMDB, minimal DML logging in 161
  - in-memory databases 153
  - incremental data transfer 136
  - longer identifiers 127
  - minimal DML logging 161
  - performance improvements for Adaptive Server replication 152
  - relaxed-durability databases 153
- suspend connection command 198, 287
- suspend log transfer command 117
- suspend route command 176, 289
- suspending
  - database connections 198
  - routes 176
- switch logins 292
- Sybase Central 86
  - creating an object in 68
  - online help 62
  - shortcut menus 65
  - starting 61
  - status bar 65
  - stopping 62
  - toolbar 65
- Sybase Control Center, Replication 30
- Sybase Failover support 97
- `sysadmin dropdb` command 107, 109, 228
- `sysadmin droprs` command 107
- `sysadmin er SSD`, command 98, 101
- `sysadmin purge_route_at_replicate` command 185
- system management tools 30
- system procedures
  - replicating 473
  - `sp_reptostandby` 452
  - `sp_setreplicate` 336
  - `sp_setrepproc` 391
  - `sp_setreptable` 329
- system tables
  - described 26
  - `rs_config` 92
  - `rs_databases` 229
  - `rs_idnames` 228

- rs\_lastcommit 224
- T**
- table replication
  - unique keys not present 311
- tables
  - creating for replication 303
  - marking for replication 338
  - materialization options 74, 75
  - procedure for replicating 306
  - subscription requirements 415
- tabs
  - help contents 62
- target-scope and replication definition-scope
  - function strings, differences 464
- target-scope functions strings 463
  - alter function string command 465
  - create function string command 465
  - drop function string command 465
- text datatype
  - changing replication for 341
  - overview of replication 336
- third-party
  - security features 19
- threads
  - DSI scheduler 52
  - RSI 52
- three-tier environment
  - preparing 79
- three-tier management solution 69
  - RMS 69
- timestamp datatypes 347
- toolbar 65
  - buttons 65
  - hiding 65
- toolbar button help 62
- tooltips 62
- topic help 62
- transactions
  - handling suspended 198
  - handling with Replication Server 49
  - modifying data in multiple data servers 57
  - skip transaction clause 224
- transfer database object ownership
  - conditions and limitations for Replication Server support 150
  - coordinating with replication definition change 149
  - Replication Server support for Adaptive Server 147
  - support for Adaptive Server 147
- triggers
  - event 80
- truncate table command
  - enabling replication of 443
  - Transact-SQL 440, 441
- truncation of Adaptive Server database logs 57
- trusted roots file 297
- two-tier environment
  - preparing 70
- two-tier management solution 69
- U**
- undesired data, removing 136
- unicode\_format configuration parameter 222
- unified\_login 283
- unique key
  - table replication without unique keys 311
- unused\_login\_expiration configuration parameter 249
- upgrade
  - LDAP 43
  - Replication Server 43
  - user authentication 43
- upgrading routes 185
- use\_batch\_markers configuration parameter 223
- use\_index 339
- use\_security\_services to 'off' parameter 293
- user
  - maintenance 73
  - RepAgent 73
- user accounts
  - configure 42
  - strings 42
- user database upgrades
  - impact of granular permissions 142
- user stored procedures
  - replicating 474
- users
  - maintenance 74, 75
  - RepAgent 74, 75
- users, displaying replication system 257
- users, ERSSD 101
- V**
- validate publication command 365, 367, 369
- validate subscription command 417, 424, 440, 444

## Index

- variables
    - function strings 48
  - verify password clause 245
  - verifying translations 377
  - version number for route 185
  - versions, replication system 17
  - viewing object properties 68
  - visual monitoring of status 66
- ## W
- warm standby
    - reducing replication definitions 462–464
    - replication definitions 462
    - subscriptions 462
  - warm standby applications
    - introduction 16
    - MSA advantages 449
    - switchover with MSA 456
    - using MSA 454
  - warm standby environment
    - configuring 72
  - where clause
    - altering 370
    - for create subscription command 418
    - operators 368
    - syntax 368
    - used in articles 368
  - wide columns 326
  - wide data 326
  - wide messages 326
  - with all tables named clause 316
  - with nowait clause 198
  - with primary table named clause 316
  - with purge clause 445
  - with replicate table named clause 316
  - without holdlock clause 421
  - without purge clause 445
- ## wizards
- Add Server 72
  - Configure Replication 72
- ## writetext 337
- ## X
- xpdl 97