

SYBASE®

Administration Guide: Volume 2

Replication Server®

12.6

DOCUMENT ID: DC32518-01-1260-02

LAST REVISED: November 2006

Copyright © 1992-2006 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.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. 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, SYBASE (logo), ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Advantage Database Server, Afaria, Answers Anywhere, Applied Meta, Applied Metacomputing, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Translator, APT-Library, ASEP, Avaki, Avaki (Arrow Design), Avaki Data Grid, AvantGo, Backup Server, BayCam, Beyond Connected, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional Logo, ClearConnect, Client-Library, Client Services, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DataWindow .NET, DB-Library, dbQueue, Dejima, Dejima Direct, Developers Workbench, DirectConnect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, EII Plus, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, ExtendedAssist, Extended Systems, ExtendedView, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, Intelligent Self-Care, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Legion, Logical Memory Manager, lRLite, M2M Anywhere, Mach Desktop, Mail Anywhere Studio, Mainframe Connect, Maintenance Express, Manage Anywhere Studio, MAP, M-Business Anywhere, M-Business Channel, M-Business Network, M-Business Suite, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, mFolio, Mirror Activator, ML Query, MobiCATS, MobileQ, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASIS, OASIS logo, ObjectConnect, ObjectCycle, OmniConnect, OmniQ, OmniSQL Access Module, OmniSQL Toolkit, OpenBridge, Open Biz, Open Business Interchange, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Pharma Anywhere, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power++, Power Through Knowledge, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Pylon, Pylon Anywhere, Pylon Application Server, Pylon Conduit, Pylon PIM Server, Pylon Pro, QAnywhere, Rapport, Relational Beans, RemoteWare, RepConnector, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RFID Anywhere, RW-DisplayLib, RW-Library, SAFE, SAFE/PRO, Sales Anywhere, Search Anywhere, SDF, Search Anywhere, Secure SQL Server, Secure SQL Toolset, Security Guardian, ShareLink, ShareSpool, SKILS, smart.partners, smart.parts, smart.script, SOA Anywhere Trademark, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase Virtual Server Architecture, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SybFlex, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TotalFix, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viafone, Viewer, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, XcelleNet, XP Server, XTNDAccess and XTNDConnect are trademarks of Sybase, Inc. or its subsidiaries. 07/06

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

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

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

CHAPTER 13	Verifying and Monitoring Replication Server	425
	Checking replication system log files for errors	426
	Verifying a replication system	426
	Monitoring Replication Server	428
	Verifying server status	429
	Displaying replication system thread status	430
	Setting and using threshold levels	432
	Monitoring partition percentages	433
CHAPTER 14	Customizing Database Operations.....	435
	Overview	435
	Working with functions, function strings, and classes	436
	Functions	437
	Summary of system functions	440
	Function strings	442
	System functions with multiple function strings	444
	Function-string classes	445
	System-provided classes	445
	Function-string inheritance	447
	Restrictions in mixed-version systems	449
	Managing function-string classes	450
	Creating a function-string class	451
	Assigning a function-string class to a database	455
	Dropping a function-string class	456
	Managing function strings	457
	Function strings and function-string classes	457
	Function-string input and output templates	457
	Using output templates	458
	Using input templates	460
	Using function-string variables	462
	Creating function strings.....	464
	Altering function strings	466
	Dropping function strings.....	467
	Restoring default function strings	469

- Creating empty function strings with the output template..... 470
- Remapping table and column names with function strings ... 470
- Defining multiple commands in a function string 470
- Using declare statements in language output templates..... 471
- Displaying function-related information 472
 - Obtaining information using the admin command..... 472
 - Obtaining information using stored procedures..... 473
- Using the default system variable 473
 - Extending default function strings 474
 - Using replicate minimal columns..... 475
- Using function strings with text, image, and rawobject datatypes 475
 - Using output writetext for rs_writetext function strings 476
 - Using output none for rs_writetext function strings 476

CHAPTER 15

Managing Warm Standby Applications 479

- Overview 480
 - How a warm standby works 480
 - Database connections in a warm standby application 481
 - Primary and replicate databases and warm standby applications
481
 - Warm standby requirements and restrictions..... 483
 - Function strings for maintaining standby databases 484
- What information is replicated? 485
 - Comparing replication methods..... 486
 - Using sp_reptostandby to enable replication 487
 - Using sp_setreptable to enable replication 490
 - Using sp_setrepproc to copy user stored procedures..... 491
 - Replicating tables with the same name but different owners 491
 - Replicating text, image, and rawobject data..... 492
 - Changing replication for the current isql session..... 493
- Setting up warm standby databases 494
 - Before you begin 494
 - Task one: Creating the logical connection 495
 - Task two: Adding the active database..... 496
 - Task three: Enabling replication for objects in the active database
497
 - Task four: Adding the standby database..... 498
- Switching the active and standby databases 506
 - Determining if a switch is necessary 507
 - Before switching active and standby databases 507
 - Internal switching steps 508
 - After switching active and standby databases 509
 - Making the switch..... 509
- Monitoring a warm standby application..... 514

Replication Server log file..... 514

Commands for monitoring warm standby applications..... 515

Setting up clients to work with the active data server 516

 Two interfaces files..... 517

 Symbolic data server name for client applications 517

 Map client data server to currently active data server 518

Altering warm standby database connections..... 518

 Altering logical connections..... 518

 Altering physical connections 521

 Dropping logical database connections..... 523

Warm standby applications using replication 524

 Warm standby application for a primary database 524

 Warm standby application for a replicate database 526

Using replication definitions and subscriptions 531

 Creating replication definitions for warm standby databases 531

 Using subscriptions with warm standby application 537

 Missing columns when you create the standby database 541

Loss detection and recovery 542

CHAPTER 16

Performance Tuning 543

Replication Server internal processing..... 543

 Threads, modules, and daemons..... 544

 Processing in the primary Replication Server 544

 Processing in the replicate Replication Server..... 549

Configuration parameters that affect performance..... 551

 Replication Server parameters that affect performance..... 551

 Connection parameters that affect performance 555

 Route parameters that affect performance..... 558

Suggestions for using tuning parameters..... 559

 Setting the amount of time SQM Writer waits 559

 Caching system tables 560

 Setting wake up intervals 560

 Sizing the SQT cache..... 560

 Controlling the number of network operations..... 561

 Controlling the number of outstanding bytes..... 562

 Controlling the number of commands the RepAgent executor can process..... 562

 Specifying the number of stable queue segments allocated . 563

 Selecting disk partitions for stable queues 563

 Making SMP more effective 563

 Specifying the number of transactions in a group 564

Using parallel DSI threads 566

 Benefits and risks 566

 Parallel DSI parameters 567

- Components of parallel DSI 570
- Processing transactions with parallel DSI threads 570
- Transaction serialization methods 572
- Partitioning rules: reducing contention and increasing parallelism
576
- Resolving conflicting updates 581
- Configuring parallel DSI for optimal performance 587
- Parallel DSI and the `rs_origin_commit_time` system variable 591
- Using multiprocessor platforms 592
 - Enabling multiprocessor support 593
 - Monitoring thread status 593
 - Monitoring performance 593
- Allocating queue segments 593
 - Choosing disk allocations 595
 - Dropping hints and partitions 596

CHAPTER 17 Using Counters to Monitor Performance 597

- Introduction 597
- Modules and counters: an overview 598
 - Modules 598
 - Counters 599
- Enabling sampling and flushing 600
 - Enabling sampling of non-intrusive counters 601
 - Enabling sampling of intrusive counters 602
 - Enabling flushing 602
- Configuring modules, connections, and routes 603
 - Flushing for all modules 603
 - Flushing for multithreaded modules 604
- Viewing information about the counters 605
- Viewing current counter values 606
- Viewing values flushed to the RSSD 607
- Resetting counters 608
- dSTATS daemon thread 608

CHAPTER 18 Handling Errors and Exceptions 609

- General error handling 609
- Error log files 610
 - Replication Server error log 610
 - RepAgent error log messages 613
- Data server error handling 615
 - Creating an error class 615
 - Initializing a new error class 616
 - Dropping an error class 616

Changing the primary Replication Server for an error class..	617
Displaying error class information	618
Assigning actions to data server errors	618
Displaying assigned actions for error numbers	619
Exceptions handling	620
Handling failed transactions	620
Accessing the exceptions log	622
Deleting transactions from the exceptions log.....	624
DSI duplicate detection	625
Duplicate detection for system transactions.....	626

CHAPTER 19 Replication System Recovery..... 627

How to use recovery procedures	628
Configuring the replication system to support Sybase Failover ...	628
Overview	629
Enabling Failover support in Replication Server	629
Configuring the replication system to prevent data loss.....	633
Save interval for recovery.....	633
Backing up the RSSDs.....	636
Creating coordinated dumps	636
Recovering from partition loss or failure.....	637
Procedure for recovering from partition loss or failure	638
Message recovery from off-line database logs.....	640
Message recovery from the online database log.....	642
Recovering from truncated primary database logs	642
Truncated message recovery from the database log	643
Recovering from primary database failures	645
Loading from coordinated dumps.....	646
Loading a primary database from dumps.....	647
Recovering from RSSD failure	648
Recovering an RSSD from dumps	649
Basic RSSD recovery procedure.....	650
Subscription comparison procedure.....	652
Subscription re-creation procedure	659
Deintegration/reintegration procedure.....	662
Recovery support tasks.....	663
Rebuilding stable queues	663

APPENDIX A Asynchronous Procedures 677

Overview	677
Logging replicated stored procedures	678
Logging replicated stored restrictions.....	678
Mixed-mode transactions	679

- Applied stored procedures 679
- Request stored procedures 680
- Asynchronous stored procedure prerequisites 681
- Steps for implementing an applied stored procedure 682
 - Warning conditions 684
- Steps for implementing a request stored procedure 686
- Specifying stored procedures and tables for replication 688
- Managing user-defined functions 689
 - Creating a user-defined function 689
 - Adding parameters to a user-defined function 690
 - Dropping a user-defined function 691
 - Mapping to a different stored procedure name 691
 - Specifying a nonunique name for a user-defined function 693

APPENDIX B

- LTM for SQL Server 695**
 - Overview 696
 - Data flow for replication systems with LTMs 696
 - LTM processing 697
 - LTM processing data flow 697
 - SQL Server LTM executable program 701
 - Shutting down an LTM 701
 - Checking log files for errors 701
 - Configuring and maintaining the LTM 702
 - Adding a Replication Server 703
 - Preparing databases for replication 703
 - Interfaces file 704
 - LTM configuration file 705
 - Replication Server login name and password for the LTM 706
 - LTM login name and password 706
 - SQL Server login name and password 707
 - Enabling and disabling password encryption 707
 - Suspending and resuming log transfer 708
 - Suspending LTMs 708
 - Resuming LTMs 709
 - Modifying replication systems with LTMs 709
 - Configuring Replication Servers to manage primary tables .. 710
 - Changing replicate databases to primary databases 710
 - Changing primary databases to replicate databases 711
 - LTM error log information 712
 - LTM message types 713

APPENDIX C

- High Availability on Sun Cluster 2.2 717**
 - Introduction 717

Terminology	718
Technology overview	719
Configuring Replication Server for high availability	720
Configuring Sun Cluster for HA	720
Installing Replication Server for HA	721
Installing Replication Server as a data service	722
Administering Replication Server as a data service	725
Data service start/shutdown	725
Logs	725
Glossary	727
Index	745



Verifying and Monitoring Replication Server

This chapter describes checking error logs, verifying that the components of a replication system are running, and monitoring the status of system components and processes.

Name	Page
Checking replication system log files for errors	426
Verifying a replication system	426
Monitoring Replication Server	428
Setting and using threshold levels	432

The replication system includes data servers and Replication Servers. It may also include replication agents for heterogeneous data servers or Log Transfer Managers for SQL Server. The replication agent for Adaptive Server is RepAgent, an Adaptive Server thread.

Note If you are using a replication agent for a heterogeneous data server, refer to replication agent documentation for your data server for information about troubleshooting your replication agent.

In a fully operational replication system, all data servers, Replication Servers, replication agents, and their internal threads and other components are running. This chapter tells you how to perform basic troubleshooting tasks on the replication system, including:

- 1 Checking error logs for status and error messages.
- 2 Logging in to system servers and checking that all threads are functioning, routes and connections are in place, and the interfaces file information is correct.

This chapter also describes how you can monitor Replication Server and its threads and check partition threshold levels.

Refer to the *Replication Server Troubleshooting Guide* for detailed information about monitoring and troubleshooting Replication Server.

Checking replication system log files for errors

The Replication Server records status and error messages, including internal errors, in the Replication Server error log file. Use the `admin log_name` command to display the path to the current log file. The default name for the log file is *repserver.log*. You can change the default name by executing `repserver` with the `-E` option and specifying the new log file name.

You can check the *repserver.log* files for any error messages by using Sybase Central. You can also invoke shell scripts based on errors reported in those logs in Sybase Central.

RSM

See “Viewing the Error Log” in Replication Server’s plug-in help for instructions on checking the information in the error logs in Sybase Central.

Internal errors are those where the only action available to Replication Server is to dump the stack and exit. For diagnostic purposes, Replication Server prints a trace of its execution stack in the log and leaves a record of its state when the error occurred.

Messages continue to accumulate in the error log files until you remove them. For this reason, you may choose to truncate the log files when the Replication Server is shut down. You can also close the Replication Server log file and begin a new log file by using the `admin set_log_name` command.

The Replication Server log file contains messages generated during the execution of asynchronous commands, such as `create subscription` and `create route`, which continue processing after the commands complete. While you are executing asynchronous commands, pay special attention to the log files for the Replication Servers affected by the procedure.

If a log file is unavailable, important error information is written to the standard error output file, which you can display on a terminal or redirect to a file.

Verifying a replication system

You need to verify that the entire replication system is working when you are about to create replication definitions or subscriptions or when you are performing diagnostics on your system. If you encounter errors, verifying your system allows you to rule out the possibility that threads or components are not running or that routes and connections are not properly set up.

RSM

See “Viewing the status of a server or folder” and the other topics in “Using the Topology View” in Replication Server’s plug-in help for information about verifying a replication system using Sybase Central.

To make sure that Replication Server threads are running, you can execute `admin who_is_down`, which displays only those threads that are not running. Alternatively, execute `admin who` to display information about all threads. If no threads are down, you can confirm that the replication system is working by checking the following:

- 1 Verify that replication system servers and replication agents are running and available.

At the primary site, log in to these servers:

- Data server with the primary database and its replication agent
If you are using Adaptive Server, execute `sp_help_rep_agent` at Adaptive Server to display status information for RepAgent thread.
- Replication Server managing the primary database
- RSSD (and its replication agent) for the primary Replication Server
If you are using Adaptive Server, execute `sp_help_rep_agent` at Adaptive Server to display status information for RepAgent thread.

At replicate sites, log in to these servers:

- Data servers with replicate databases and, if request functions are executed at these databases, their replication agents
If you are using Adaptive Server, execute `sp_help_rep_agent` at Adaptive Server to display status information for RepAgent thread.
- Replication Servers managing replicate databases
- RSSDs (and their replication agents) for replicate Replication Servers
If you are using Adaptive Server, execute `sp_help_rep_agent` at Adaptive Server to display status information for RepAgent thread.

- 2 Use the `admin show_connections` command at Replication Server to verify that these routes and connections are in place:

- Routes from the primary Replication Server to each replicate Replication Server
- Database connection between the primary Replication Server and the primary database

- Route from a replicate Replication Server to the primary Replication Server, if the replicate Replication Server manages a replicate database in which request functions are executed
 - Database connections between each replicate Replication Server and its replicate database
- 3 Verify the accuracy of entries in the interfaces file.

When creating subscriptions, be sure an entry for the primary data server exists in the interfaces file for the replicate Replication Server. (If you are using atomic or non-atomic materialization, the replicate Replication Server retrieves initial rows through a direct connection to the primary data server.)

- 4 Use the admin who command to verify that these Replication Server threads are running:
- Data Server Interface (DSI)
 - Replication Server Interface (RSI)
 - Distributor (DIST)
 - Stable Queue Manager (SQM)
 - Stable Queue Transaction interface (SQT)
 - RepAgent User

For detailed information about monitoring Replication Server threads, refer to “Displaying replication system thread status” on page 430.

Monitoring Replication Server

While the replication system is in operation, you may need to monitor its components and processes. This section describes how to:

- Monitor replication system servers
- Monitor DSI, RSI, and other thread status
- Use system information commands to obtain information about various aspects of the Replication Server.

Verifying server status

You can verify the status of your servers with these methods:

- Use Sybase Central.
RSM – See “Viewing the status of a server or folder” in Replication Server’s plug-in help for instructions on verifying the status of your servers using Sybase Central.
- Use `isql` to log in to each server. If the login succeeds, you know that the server is running.
- Create a script that logs in to and displays the status of each Adaptive Server and its RepAgent thread, other replication agent (if any), and Replication Server. Make sure all servers in the script are included in the interfaces file.

If a login fails, it may be caused by one of the following problems:

Problem: You typed an incorrect name, or the interfaces file you are using does not have an entry for the server.

```
DB-LIBRARY error:
  Server name not found in interface file.
```

Problem: The server is running, but you specified an incorrect login name or password.

```
DB-LIBRARY error:
  Login incorrect.
```

Problem: The server is not running.

```
Operating-system error:
  Invalid argument
DB-LIBRARY error:
  Unable to connect: SQL Server is unavailable
  or does not exist.
```

Problem: The interfaces file cannot be found.

```
Operating-system error:
  No such file or directory
DB-LIBRARY error:
  Could not open interface file.
```

Problem: The interfaces file exists, but you do not have permission to access it.

```
Operating-system error:
  Permission denied
DB-LIBRARY error:
```

Could not open interface file

Displaying replication system thread status

You can monitor general information on current Replication Server threads. Table 13-1 describes threads that apply to database connections and routes and the admin who command you use to monitor them.

RSM See “Viewing thread status” in Replication Server’s plug-in help for Sybase Central instructions on monitoring threads.

Table 13-1: Monitoring Replication Server threads

Replication Server thread	Command
Distributor (DIST) – uses SQT and SQM to read transactions from the inbound queue.	admin who, dist
Data Server Interface (DSI) – submits transactions to data server.	admin who, dsi
REP AGENT or LTM USER – verifies that transactions from the data server are valid and writes them to the inbound queue.	admin who Note Use sp_who or sp_help_rep_agent to display status of RepAgent thread at Adaptive Server.
Replication Server Interface (RSI) – logs in to each destination Replication Server and transfers commands from the stable queue to the destination server.	admin who, rsi
Stable Queue Manager (SQM) – manages Replication Server stable queues.	admin who, sqm
Stable Queue Transaction interface (SQT) – reads transactions in a queue and passes them to the SQT reader.	admin who, sqt

Refer to “admin who” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for details on the admin who command. Refer to the *Replication Server Troubleshooting Guide* to interpret the command output for troubleshooting purposes.

Using system information commands

In addition to admin who, Replication Server offers other admin commands to assist you in monitoring Replication Server.

These commands are listed in Table 13-2. Refer to Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for details on each command.

RSM In Sybase Central, you can find additional information about monitoring your replication system in “Monitoring a Replication System” in Replication Server’s plug-in help.

Table 13-2: Overview of system information commands

Command	Description
admin disk_space	Displays utilization of disk partitions accessed by the Replication Server.
admin echo	Determines if the local Replication Server is running.
admin get_generation	Retrieves the generation number for a primary database, used in recovery operations.
admin health	Displays the overall status of the Replication Server.
admin log_name	Displays the path to the current log file.
admin logical_status	Displays the status of logical database connections, used in warm standby applications.
admin pid	Displays the process ID of the Replication Server.
admin quiesce_check	Determines if the queues in the Replication Server have been quiesced.
admin quiesce_force_rsi	Determines whether a Replication Server is quiescent. Also forces Replication Server to deliver outbound messages.
admin rssid_name	Displays the names of the data server and database for the RSSD.
admin security_property	Displays security features of network-based security systems supported by Replication Server.
admin security_setting	Displays network-based security settings of a particular target server.
admin set_log_name	Closes the existing Replication Server log file and opens a new log file.
admin show_connections	Displays information about all connections and routes to and from Replication Server.
admin show_function_classes	Displays the names of existing function-string classes and their parent classes and indicates the number of levels of inheritance.
admin show_route_version	Displays the version number of routes that originate at Replication Server and routes that terminate at Replication Server.
admin show_site_version	Displays the site version of Replication Server.
admin sqm_readers	Displays information about threads that are reading the inbound queue.
admin statistics, flush_status	Displays flushing status for all counters.
admin statistics, md	Displays statistics about message delivery and counters.
admin statistics, mem	Displays statistics about memory utilization.
admin statistics, reset	Resets the message delivery statistics.
admin stats_config_connection	Controls flushing of metrics to the RSSD for connections.
admin stats_config_module	Enables or disables flushing of metrics to the RSSD for one or all modules.
admin stats_config_route	Controls flushing of metrics to the RSSD for routes.
admin stats_intrusive_counter	Enables or disables sampling for intrusive counters.

Command	Description
admin version	Displays which version of the Replication Server you are running, representing the software version.
admin who	Displays information about all threads in the Replication Server.
admin who, dsi	Displays information about DSI threads that connect to a data server.
admin who, rsi	Displays information about RSI threads that connect to other Replication Servers.
admin who, sqm	Displays information about all queues managed by the SQM.
admin who, sqt	Displays information about all queues managed by the SQT.
admin who_is_down	Displays the same information as admin who, but only about threads that are down.
admin who_is_up	Displays the same information as admin who, but only about threads that are running.

Setting and using threshold levels

Stable queue partitions fill up when a Replication Server is receiving more messages than it is sending. For example, if a network is down between a primary site and a replicate site, the Replication Server at the primary site queues up the undeliverable messages. When the network returns to service, the messages can be delivered, and then deleted from the primary Replication Server's partitions.

If a partition becomes completely full, senders cannot deliver their messages to the Replication Server, and messages begin to back up in the partitions at previous sites and in the transaction logs for primary databases.

Warning! If the situation is not corrected, RepAgent is unable to update the secondary truncation point in the database log, and the transaction log fills. Clients are then unable to execute transactions at the primary database.

You can configure Replication Server to warn when partitions become too full by setting three rows in the `rs_config` system table: `sqm_warning_thr1`, `sqm_warning_thr2`, and `sqm_warning_thr_ind`. These parameters are described in Figure 16-2 on page 550.

RSM

See “Managing partition events” in Replication Server’s plug-in help.

Monitoring partition percentages

Replication Server operates on 1MB partition segments. Whenever it allocates or deallocates a partition segment, it calculates these statistics:

- Percentage of total partition segments in use
- Percentage of total partition segments in use by the affected stable queue

If the percentage of partition segments in use rises above the percentage specified by `sqm_warning_thr1` or `sqm_warning_thr2`, a message like the following is written to the log file:

```
WARNING: Stable Storage Use is Above <threshold> percent
```

If you see this message often, you may need to add partitions to the Replication Server or correct a recurring failure that causes the queues to fill.

When the first percentage drops below the percentage specified by `sqm_warning_thr1` or `sqm_warning_thr2`, a message like the following is written to the log file to note that the condition that caused the original warning no longer exists:

```
WARNING CANCEL: Stable Storage Use is Below <threshold>
percent
```

The percentage of total partition segments in use by the affected stable queue triggers the following warning message when the percentage of the total space used by a single stable queue exceeds the percentage specified by `sqm_warning_thr_ind`:

```
WARNING: Stable Storage Use by <queue name> is Above
<threshold> percent
```

This warning alerts you to problems that cause a particular stable queue to fill until it is using a disproportionate share of the total partition space. For example, if a route is suspended for a length of time, its stable queue may fill until it occupies enough partition space to trigger a warning.

When the percentage of the total partition space used by a stable queue drops below the `sqm_warning_thr_ind` percentage, Replication Server writes a cancel message like the following to the log file:

```
WARNING CANCEL: Stable Storage Use by <queue name> is
Below <threshold> percent.
```


Customizing Database Operations

This chapter explains how you can create and alter functions, function strings, and function-string classes to allow replication definitions to work with database servers other than Adaptive Server.

Name	Page
Overview	435
Working with functions, function strings, and classes	436
Function-string classes	445
Managing function-string classes	450
Managing function strings	457
Displaying function-related information	472
Using the default system variable	473
Using function strings with text, image, and rawobject datatypes	475

Overview

Replication Server translates commands from the primary database into Replication Server functions that represent data server operations such as insert, delete, select, begin transaction, and so on. It distributes these functions to remote Replication Servers in the system, where they execute those operations in remote databases.

The primary Replication Server distributes functions in the same format regardless of the type of data server that actually updates the replicated data. Functions are not database-specific. They include all the data needed to perform the operation, but they do not specify the syntax needed to complete the operation at the destination data server.

The remote Replication Server converts functions to commands specific to the destination data servers where they are executed. A function string contains the database-specific instructions for executing a function. The replicate Replication Server managing a database uses an appropriate function string to map the function to a set of instructions for the data server. For example, the function string for the `rs_insert` function provides the actual language to be applied in a replicate database.

This separation between functions and data server commands lets you maintain replicated data among heterogeneous data servers. Replication Server allows you to customize function strings, specifying how Replication Server functions map to SQL commands. You can create function strings if you require customized data server operations. You customize replicated data applications by changing the way operations are performed at the destination database.

Function strings are grouped into function-string classes, so you can group mappings of functions to commands according to data server. Replication Server provides function-string classes for Adaptive Server Enterprise, Oracle, Informix, Microsoft SQL Server, Adaptive Server Anywhere, IMS, VSAM, and DB2 databases. You can create new derived function-string classes in which you customize certain function strings and inherit all others from these or other classes. You can also create entirely new classes in which you create all new function strings.

You may also need to create function strings for replicated functions, which allow you to execute stored procedures on remote databases. You must create a function string for any replicated function for which Replication Server does not automatically generate a function string in the function-string class used by the destination database.

Working with functions, function strings, and classes

You can work with functions and function strings to customize database operations in any of these ways:

- Create a new function-string class for use with a specific type of database, and customize some or all of the function strings. See “Managing function-string classes” on page 450 for detailed information.
- For atomic materialization, use a function from a function-string class associated with the primary database connection, not a function from the function-string class associated with the replicate database connection.

- Alter function strings for the system-provided function-string class, `rs_sqlserver_function_class`. See “Managing function strings” on page 457 for detailed information.
- Create a function-string class that inherits, either directly or indirectly, function strings from the system-provided function-string class `rs_default_function_class`.
- Use the system-provided function-string classes for non-Sybase data servers: `rs_db2_function_class`, `rs_informix_function_class`, `rs_mss_function_class`, or `rs_oracle_function_class`. See “Translating datatypes using HDS” on page 319 for detailed information on datatype translations using the heterogeneous datatype support (HDS) feature.

This section provides an overview of functions, function strings, and function-string classes. The following sections include a summary of the system functions, procedures, and guidelines for managing function strings and function-string classes. They also summarize commands for displaying information about the function strings and classes in the replication system.

You can work with functions, function strings, and classes using Sybase Central or RCL commands. This chapter describes procedures and RCL commands that you enter at the command line using `isql`.

RSM

For information about using Sybase Central, see Replication Server’s plug-in help.

Refer to Chapter 4, “Replication Server System Functions,” in the *Replication Server Reference Manual* for more information about the system functions.

Functions

Replication Server uses two major types of functions:

- System functions
- User-defined functions

You can create custom function strings for either type of function, depending on your needs.

See “Managing function strings” on page 457 for more information about when to customize function strings.

System functions

System functions represent data server operations whose function strings are supplied by Replication Server or are available when you install a new database on the replication system. Unless your application requires it, you do not need to customize function strings for system functions. The system-provided class generates them for you.

System functions include:

- Functions that represent data-manipulation operations such as insert, update, delete, select, and select with holdlock.

These system functions have replication-definition scope. See “Function scope” on page 439 for details.

- Functions that represent transaction-control directives. These functions include operations such as begin transaction and commit transaction.

These system functions have function-string-class scope. See “Function scope” on page 439 for details.

See “Summary of system functions” on page 440 for more information about each type of system function.

User-defined functions

User-defined functions allow you to use Replication Server to distribute replicated stored procedures between sites in the replication system. You must create function strings for user-defined functions unless you use a function-string class that directly or indirectly inherits function strings from `rs_default_function_class`. User-defined functions include:

- Functions that are used in replicating stored procedures associated with function replication definitions. Replication Server automatically creates a user-defined function of this type when you create a function-replication definition.

See Chapter 10, “Managing Replicated Functions” for details about function-replication definitions and replicated stored procedures.

- Functions that are used in replicating stored procedures associated with table-replication definitions. You create and maintain user-defined functions of this type yourself.

For details about replicated stored procedures that use table-replication definitions, see Appendix A, “Asynchronous Procedures.”

User-defined functions have replication-definition scope. See “Function scope” on page 439 for details.

Any function string that you create for a user-defined function should be created at the primary Replication Server, where the replication definition was created. If you are using function replication definitions, see also “Implementing an applied function” on page 338 or “Implementing a request function” on page 341.

Function scope

The scope of a function defines the object to which the function applies: either to a replication definition or to a function-string class. Knowing a function’s scope is important for determining where to customize a function string: at the primary or replicate Replication Server. Functions can have one of two scopes:

- Function-string-class scope
- Replication-definition scope

Function-string-class scope

A function with function-string-class scope is defined once for the class. Functions with function-string-class scope include system functions that represent transaction-control directives (such as `rs_begin`, `rs_commit`, or `rs_marker`) and do not perform data manipulation. Function strings for user-defined functions do not have class scope.

Function strings for functions with function-string-class scope must be customized at the primary Replication Server for the function-string class. See Table 14-1 on page 440 for a list of these functions. See “Primary site for a function-string class” on page 453 for information on assigning a primary site.

Replication-definition scope

A function with replication-definition scope is defined once for a specific table-replication definition or function-replication definition—although the function may have multiple function strings.

Functions with replication-definition scope include:

- System functions that perform data-manipulation operations (such as `rs_insert`, `rs_delete`, `rs_update`, `rs_select`, `rs_select_with_lock`, and special functions used in replicating text and image data).

See Table 14-2 for a list of these functions.

- User-defined functions for table- or function-replication definitions.

System functions with replication-definition scope must be customized at the Replication Server where the replication definition was created. User-defined functions with replication-definition scope must be customized at the Replication Server where the replication definition was created.

Summary of system functions

The following tables provide a summary of the available system functions. Refer to Chapter 4, “Replication Server System Functions,” in the *Replication Server Reference Manual* for complete documentation of all of the system functions.

System functions with function-string-class scope

Table 14-1 lists the system functions with function-string-class scope. Replication Server provides default generated function strings for each system-provided class when you install the replication system.

Some functions are required for every Replication Server application, while other functions only apply in particular cases, such as warm standby applications, parallel DSI threads, or coordinated dumps.

If you use a function-string class other than the default (`rs_sqlserver_function_class`), and you are not using function-string inheritance, you must create a function-string for each system function you use that has function-string class scope.

Customize function strings for system functions with class scope at the Replication Server that is the primary site for the function-string class. See “Changing the primary site for a function-string class” on page 454 for more information about assigning or changing the primary Replication Server for a function-string class.

Table 14-1: System functions with function-string-class scope

Function name	Description
<code>rs_begin</code>	Begin a transaction.
<code>rs_check_repl</code>	Check if a table is marked for replication.
<code>rs_commit</code>	Commit a transaction.
<code>rs_dumpdb</code>	Initiate a coordinated database dump.
<code>rs_dumptran</code>	Initiate a coordinated transaction dump.

Function name	Description
rs_get_charset	Return the character set used by a data server. Sample function strings for replication into DB2 databases via Net-Gateway are installed in the Sybase release directory in <i>install/rs_db2_setup.sample</i> (UNIX systems) and <i>install/rs_2_db2.txt</i> (Windows 2000, 2003 systems).
rs_get_lastcommit	Retrieve rows from the rs_lastcommit system table.
rs_get_sortorder	Return the sort order used by a data server. Sample function strings for replication into DB2 databases via Net-Gateway are installed in the Sybase release directory in <i>install/rs_db2_setup.sample</i> (UNIX systems) and <i>install/rs_2_db2.txt</i> (Windows 2000 and 2003 systems).
rs_get_thread_seq	Return the current sequence number for the specified entry in the rs_threads system table. This function is executed only when you are using parallel DSI.
rs_get_thread_seq_noholdlock	Return the current sequence number for the specified entry in the rs_threads system table, using the noholdlock option. This function is executed only when you are using parallel DSI with isolation level 3 locking.
rs_initialize_threads	Set the sequence of each entry in the rs_threads system table to 0. This function is executed only when you are using parallel DSI.
rs_marker	Help coordinate subscription materialization. The function passes its first parameter to Replication Server as an independent command.
rs_raw_object_serialization	Replicate Java columns as serialized data.
rs_repl_off	Set replication off in Adaptive Server for a standby database connection.
rs_rollback	Roll back a transaction.
rs_set_isolation_level3	Turn on transaction isolation level 3 locking in Adaptive Server. This function is executed only when you are using parallel DSI with isolation level 3 locking.
rs_set_proxy	Assume the permissions, login name, and server user ID of the user.
rs_thread_check_lock	Determines whether or not the DSI executor thread is holding a lock that blocks a replicate database process.
rs_triggers_reset	Set triggers off in Adaptive Server for a standby database connection.
rs_trunc_reset	Reset the secondary truncation point in warm standby databases. This function is executed only when you create a warm standby database or when you switch to a standby database.
rs_trunc_set	Set the secondary truncation point in warm standby databases. This function is executed only when you create a warm standby database or when you switch to a standby database.
rs_update_threads	Update the sequence number for the specified entry in the rs_threads table. This function is executed only when you are using parallel DSI.
rs_usedb	Change the database context.

System functions with replication-definition scope

Table 14-2 lists the system functions with replication-definition scope. Replication Server provides default function strings for each system-provided class when you create a replication definition.

Some functions are required for every Replication Server application, while other functions only apply in particular cases, such as replication of text and image datatypes, parallel DSI threads, or performing subscription materialization or dematerialization.

Customize function strings for a system functions with replication-definition scope at the Replication Server where the replication definition was created.

Table 14-2: System functions with replication definition scope

Function name	Description
rs_datarow_for_writetext	Provide an image of the data row associated with a text or image column updated with a Transact-SQL writetext command or with CT-Library or DB-Library functions.
rs_delete	Delete a row in a table.
rs_get_textptr	Retrieve the text pointer for a text, image, or rawobject column.
rs_insert	Insert a row into a table.
rs_select	Retrieve rows from a table for subscription materialization or dematerialization.
rs_select_with_lock	Retrieve subscription materialization or dematerialization rows using a holdlock.
rs_textptr_init	Allocate a text pointer for a text, image, or rawobject column.
rs_truncate	Truncate a table.
rs_update	Update a row in a table.
rs_writetext	Alter text, image, or rawobject data.

Function strings

Function strings contain instructions for executing a function in a database. These instructions may differ according to database. For example, a non-Sybase database may require different instructions and have different function strings than an Adaptive Server database.

Function strings come in two formats: language and RPC. A language-format function string contains a command, such as a SQL statement, that the data server parses. An RPC-format function string contains a remote procedure call that executes a registered procedure in an Open Server gateway application or in an Adaptive Server database. Both function-string formats can contain variables that get replaced with data values. What format a function string uses is determined by the type of data server and how you want Replication Server to interact with it. See “Using output templates” on page 458 for more information.

Function strings are grouped into function-string classes. Each database connection must be assigned a function-string class according to the type of destination database. Replication Server provides function-string classes that contain default function strings. Replication Server generates default function strings for Adaptive Server, SQL Server, DB2, Informix, Microsoft SQL Server, and Oracle function-string classes.

When you set up a replication system or add databases to the system, you should anticipate your function-string requirements and decide how you will use function-string classes and whether you need to customize function strings. See “Function-string classes” on page 445 for more information.

See “Managing function strings” on page 457 for more information about customizing function strings.

Input and output templates

Every function string uses an output template to instruct the destination database in executing the function for a specific data server.

Function strings for the `rs_select` and `rs_select_with_lock` functions use both input templates and output templates, which together perform subscription materialization and dematerialization.

You customize function strings by altering their input and output templates. You customize function strings for functions other than `rs_select` and `rs_select_with_lock` by altering only the output template. How you alter a function string depends on the function string’s format—language or RPC.

See “Function-string input and output templates” on page 457 for more information about input and output templates.

Applications for customized function strings

You can customize function strings to:

- Perform operations in any native database language (including those other than Transact-SQL) by altering function-string output templates to format the commands sent to a data server.
- Materialize and dematerialize multiple subscriptions for the same replication definition with a single function string.
- Perform the following tasks by altering output templates for existing system function strings:
 - Record auditing information.
 - Execute remote procedure calls (RPCs).
 - Replicate data into multiple replicate tables in the same database.
 - Replicate data into a replicate table with a different name, column names, or column order than the primary table.

If the replicate Replication Server is of version 11.5 or later, you can perform the same tasks more easily by creating a customized replication definition that specifies the relevant information about the replicate table. See “Creating multiple replication definitions per table” on page 271 for more information.

System functions with multiple function strings

For the class-scope system functions, each function maps to a function string within the class. Each replication-definition-scope `rs_insert`, `rs_delete`, and `rs_update` system function maps to a function string within the class for each replication definition.

You can create multiple function-string instances for the same replication definition for other system functions with replication-definition scope—`rs_select`, `rs_select_with_lock`, `rs_datarow_for_writetext`, `rs_get_textptr`, `rs_textptr_init`, and `rs_writetext`. In such cases, you must give each instance of a function string a different name. System functions that can take multiple function strings include:

- `rs_select` and `rs_select_with_lock` functions – used in subscription materialization and dematerialization when multiple subscriptions exist for the same replication definition. You can give each instance of the function string any name that is unique for the replication definition. Each instance of the function string corresponds to a where clause used in creating subscriptions for the replication definition.

- `rs_datarow_for_writetext`, `rs_get_textptr`, `rs_textptr_init`, and `rs_writetext` function each instance of the function string. You must name each instance of a function string for the text or image column specified in the replication definition.

Function-string classes

Each function string belongs to a function-string class, which groups function strings intended to be used with databases of a similar type or with similar requirements. Replication Server assigns each database connection a function-string class according to the data server of the destination database.

Replication Server applies functions to the database using the function strings from its assigned function-string class. Function-string classes contain function strings for system functions and for any user-defined functions.

You can use a function-string class on multiple databases if the function strings can execute on all of the data servers. For example, a system with several databases managed by Adaptive Server can use `rs_sqlserver_function_class` for all the databases.

You can even use a single function-string class with heterogeneous data servers, provided that the gateways that provide access to the various data servers share a common interface.

System-provided classes

Several function-string classes are provided with Replication Server. These are called **system-provided classes**.

- `rs_sqlserver_function_class` – default Adaptive Server function strings are provided for this class. The default function strings in `rs_sqlserver_function_class` are identical to those in `rs_default_function_class`. `rs_sqlserver_function_class` is assigned by default to Adaptive Server databases you add to the replication system using `rs_init`.

You can customize function strings for this class. However, this class cannot participate in function-string class inheritance. In most cases, using derived classes that specify `rs_default_function_class` as a parent class is preferable to using `rs_sqlserver_function_class` directly.

- `rs_default_function_class` – default Adaptive Server function strings are provided for this class. The default function strings in `rs_sqlserver_function_class` are identical to those in `rs_default_function_class`.

You cannot customize function strings for this class. However, this class can participate in function-string class inheritance. In most cases, using derived classes that specify `rs_default_function_class` as a parent class is preferable to using `rs_default_function_class` directly.

Note The system-provided function-string classes `rs_default_function_class` and `rs_sqlserver_function_class` contain default function strings for all system functions except `rs_dumpdb` and `rs_dumprtran`. If you need to use function strings for these functions you must create them yourself in a derived class or in `rs_sqlserver_function_class`.

- `rs_db2_function_class` – DB2-specific function strings are provided for this class. See “Creating class-level translations” on page 322 for more information about using this class.

To allow `rs_db2_function_class` and other function-string classes to work, issue the following commands:

```
alter connection to dataserver.database
set dsi_sql_data_style to 'db2'
alter connection to dataserver.database
set dsi_cmd_separator to ';' ;
```

The `rs_writetext` function string of `rs_db2_function_class` was changed to “output none.” `rs_db2_function_class` does not support replication of text or image data. To achieve this functionality, customize the `rs_writetext` function string using the RPC method through a gateway.

You cannot customize function strings for this class. If you require DB2 function strings, using derived classes that specify `rs_db2_function_class` as a parent class is preferable, in most cases, to using this class directly.

- `rs_informix_function_class` – Informix function strings are provided for this class. You cannot customize function strings for this class. See “Creating class-level translations” on page 322 for more information about using this class.

- `rs_mss_function_class` – Microsoft SQL Server function strings are provided for this class. You cannot customize function strings for this class. See “Creating class-level translations” on page 322 for more information about using this class.
- `rs_oracle_function_class` – Oracle function strings are provided for this class. You cannot customize function strings for this class. See “Creating class-level translations” on page 322 for more information about using this class.

Table 14-1 on page 440 illustrates function-string inheritance relationships for these and other classes.

Function-string inheritance

The ability to share function-string definitions among classes by creating relationships between classes is called **function-string inheritance**.

Using function-string inheritance in general, and inheriting from system-provided classes in particular, provides both administrative and upgrade benefits to Replication System Administrators. Using classes that inherit from system-provided classes, you alter only the function strings you want to customize and inherit all others.

If you use classes that do not inherit from system-provided classes, you must create all function strings yourself, and add new function strings whenever you create a new table or function replication definition.

A class that inherits function strings from a parent class is called a **derived class**. A class from which a derived class inherits function strings is called the **parent class** of the derived class. Generally, you create a derived class in order to customize certain function strings and inherit all others from the parent class.

A class that does not inherit function strings from any parent class is called a **base class**. The system-provided classes `rs_default_function_class` and `rs_db2_function_class`, and any additional classes you create that do not inherit function strings from a parent class, are base classes. The system-provided classes `rs_informix_function_class`, `rs_mssql_function_class`, `rs_oracle_function_class` are derived from `rs_default_function_class`.

A parent class can have multiple derived classes, while a derived class can have only one parent class. A derived class can also serve as the parent class for one or more derived classes. A set of derived classes of any number of levels stemming from the same base class is called a **class tree**.

The system-provided classes `rs_default_function_class` and `rs_db2_function_class` can serve as parent classes for derived classes. However, they cannot become derived classes of other parent classes.

The system-provided class `rs_sqlserver_function_class` cannot serve as a parent class or become a derived class.

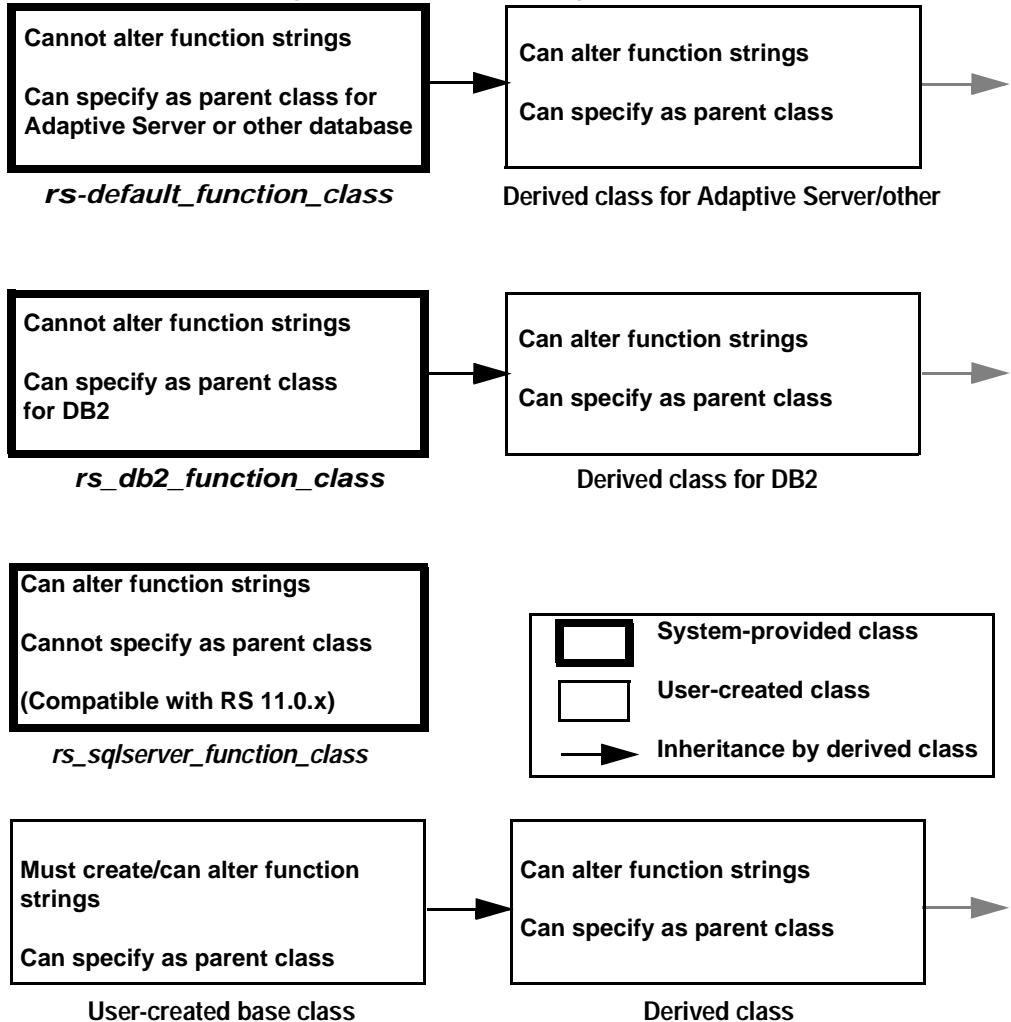
A base class that you have created can be modified to become a derived class, or it can be designated as the parent class for a derived class. A derived class can be modified to inherit function strings from a different parent class, or it can be detached from a parent class and become a base class.

For every base class that you create, you must provide function strings for the functions that Replication Server invokes in each database to which the class is assigned. If you assign a function-string class to a database when some of the function strings for system functions are missing, the DSI reports an error when Replication Server tries to apply the function string, and suspends the database connection.

Circular function-string inheritance relationships are disallowed. That is, a parent class cannot be modified to inherit function strings from one of its own derived classes or from a derived class of one of these derived classes.

Function-string class relationships are illustrated in Figure 14-1.

Figure 14-1: Function-string class relationships



Restrictions in mixed-version systems

In a mixed-version system, only Replication Servers of version 11.5 or later can work with classes that participate in function-string inheritance.

Any class whose primary site is Replication Server version 11.0.x cannot participate in function-string inheritance. If you want to alter such a class to become a derived class or use it as a parent class, you must move that class to a primary site that is Replication Server version 11.5 or later. Then you can alter the class relationships as desired and assign the class or its derived classes to connections managed by Replication Server version 11.5 or later.

A base class that you created in Replication Server version 11.5 or later and that does not participate in function-string inheritance can be assigned to connections managed by any Replication Server in the replication system. If it is not assigned to any databases managed by Replication Server version 11.5 or later, then you can use the `move primary` command to assign it to a primary site managed by Replication Server version 11.0.x.

Refer to the release bulletin for more information about compatibility between Replication Servers.

Note For compatibility with Replication Servers of version 11.0.x, you may need to continue to customize function strings in `rs_sqlserver_function_class`. However, for databases managed by Replication Servers version 11.5 or later, using function-string inheritance and customizing function strings only in derived classes is encouraged.

Managing function-string classes

When you create or customize a function string, you specify which class it belongs to. If you want to create and use customized function strings, you can:

- Create a derived function-string class that inherits function strings from `rs_default_function_class`, `rs_db2_function_class`, or another parent class. Then, in the derived class, create only the function strings that you are interested in overriding.

Note You cannot alter, add to, delete, or change any of the function-string classes for non-Sybase data servers.

- Create a new function-string class and create function strings for all functions.

- Customize function strings in `rs_sqlserver_function_class`. See “Managing function strings” on page 457 for information on this option.

Before you create customized function strings, you should decide in advance which of these approaches to take and set up your classes accordingly.

Generally, it is preferable to customize function strings in derived classes rather than to customize function strings in the class

`rs_sqlserver_function_class`. You must be using Replication Server version 11.5 or later in order to create and deploy a derived function-string class that inherits function strings from other classes.

Creating a function-string class

If function strings in an existing class do not serve your needs for particular database connections, and customizing function strings in an existing class is not feasible, you can create a new class in which to create the function strings you need. You can either:

- Create a derived class, one that inherits function strings from an existing parent class.
- Create a base class, one that does not inherit function strings from another class.

RSM

For instructions on creating a derived or base function-string class in Sybase Central, see “Creating function-string classes” in Replication Server’s plug-in help.

To create a derived or base function-string class and begin using it for a database connection using RCL commands, follow these steps:

- 1 Create the function-string class with the `create function string class` command, using the syntax appropriate for your task. See:
 - “Creating a derived class” on page 452, or
 - “Creating a base class” on page 453.

The name of the new class must conform to the rules for identifiers provided in “Identifiers” in Chapter 2, “Topics,” in the *Replication Server Reference Manual*.

- 2 Create function strings for the new class with the `create function string` command, described in “Creating function strings” on page 464.

- If you are creating a derived class, you need create only the function strings that you want to override and inherit all others from the specified parent class.
 - The class `rs_default_function_class` does not contain default function strings for the `rs_dumpdb` and `rs_dumpran` functions. If you require them in a derived class that inherits from `rs_default_function_class`, you must create them. See “System-provided classes” on page 445 for more information.
 - If you are creating a base class, you must create all the necessary function strings for the class.
- 3 If you are preparing a new function-string class for an existing database connection, you must suspend the connection before you can use the new class. See “Suspending database connections” on page 177 for details.
 - 4 Create or alter the database connection to use the new class. See “Assigning a function-string class to a database” on page 455.
 - 5 If you altered an existing database connection to use the new class, resume the connection. See “Suspending database connections” on page 177 for details.

Creating a derived class

To create a derived function-string class that inherits function strings from a parent class, enter a command like this at the primary site of the parent:

```
create function string class
  sqlserver_derived_class
  set parent to rs_default_function_class
```

In this example, the new class `sqlserver_derived_class` inherits function strings from the system-provided class `rs_default_function_class`. You can then create function strings that override some of the inherited function strings.

You can specify as the parent class any existing class whose primary site runs Replication Server version 11.5 or later. However, you cannot specify as a parent class the system-provided class `rs_sqlserver_function_class`. You also cannot specify a parent class that would result in circular inheritance. See “Function-string inheritance” on page 447 for details.

If the parent class is `rs_default_function_class` or a function-string class for a non-Sybase data server, you can enter this command at any Replication Server with routes to the other Replication Servers where the new class will be used. This site is the primary site for the derived class and any new classes derived from it.

If the parent class is a user-created class, enter this command in the Replication Server that is the primary site for the parent class. This site is the primary site for all classes derived from the parent class.

Creating a base class

To create a base function-string class, one which does not inherit function strings from a parent class, enter a command like this:

```
create function string class base_class
```

In this example, the new class `base_class` does not inherit function strings from a parent class.

Enter this command at any Replication Server that has routes to the other Replication Servers where the new class will be used. This site then becomes the primary site for the class and for any derived classes for which this class serves as the parent class.

A base class can be used as a parent class for a derived class or can be modified to become a derived class.

For every base class that you create, you must provide function strings for the functions that Replication Server invokes in each database to which the class is assigned.

If you create a base class and then alter it so it becomes a derived class before actually using it with database connections, you do not have to create all the function strings.

Primary site for a function-string class

Although most function strings are executed in replicate databases, you execute the `create function string class` command in a Replication Server, usually a primary Replication Server, that has routes to all sites where the function-string class is to be used. This command designates that Replication Server as the primary site for the class. Function-string classes are replicated via routes, along with other replication system data.

You can only create or alter function strings that have class scope at the primary site for a class. Function strings with replication-definition scope must be created or altered at the primary site for the replication definition.

By default, the class `rs_sqlserver_function_class` does not have a primary site. To alter class-scope function strings for this class, you must first designate a Replication Server as a primary site for the class. To specify a site for this function-string class, execute the following command at the Replication Server that is to be the primary site:

```
create function string class
rs_sqlserver_function_class
```

After you have executed this command, you can use the `move primary` command to make further changes to the primary site for the function-string class.

Changing the primary site for a function-string class

Use the `move primary` command or Sybase Central to change the primary Replication Server for a function-string class. For example, you may need to change the primary site from one Replication Server to another so that function strings can be distributed through a new routing configuration. The new primary site must include routes to all Replication Servers where the function-string class will be used.

If you move a base class, all classes derived from that class move with it.

You cannot move the primary site for a derived class unless its parent class is a default function-string class.

RSM

For instructions on changing the primary site in Sybase Central, see “Moving a function-string class to a different primary site” in Replication Server’s plug-in help.

Execute `move primary` at the Replication Server that you want to designate as the new primary site for the function-string class.

For example, the following command changes the primary site for the `sqlserver2_function_class` function-string class to the `SYDNEY_RS` Replication Server, where the command is entered:

```
move primary of function string class
sqlserver2_function_class
to SYDNEY_RS
```


If the class `rs_sqlserver_function_class` has not yet been assigned a primary site, you cannot use the `move primary` command to assign one. You must use the `create function string class` command to first designate a primary site for that class. See “Changing the primary site for a function-string class” on page 454 for details.

Assigning a function-string class to a database

You can assign a function-string class to a database connection in Sybase Central or with the `create connection` or `alter connection` commands, executed in the Replication Server that manages the database. When you add a database connection using the `rs_init` program, the class `rs_sqlserver_function_class` is assigned to the database by default.

RSM

For instructions on assigning a function-string class to a database in Sybase Central, see “Assigning function-string classes to database connections” in Replication Server’s plug-in help.

You must suspend the connection to the database before you alter the function-string class that is assigned to the database. The `set function string class` clause of `create connection` and `alter connection` specifies the name of the function-string class to use with the database.

Before you can assign a function-string class to a database connection:

- The function-string class you specify must already exist and be available to the Replication Server. See “Creating a function-string class” on page 451 for more information.
- All necessary function strings must be created in the class. See “Creating function strings” on page 464 for details.

See “Creating database connections” on page 174 and “Altering database connections” on page 177 for more information about using the `create connection` and `alter connection` commands. Also refer to reference pages for these commands in the *Replication Server Reference Manual*.

Refer to the Replication Server installation and configuration guides for your platform for more information about `rs_init`.

Example for creating new connection

The following command creates a connection to the `pubs2` database managed by the `TOKYO_DS` data server:

```
create connection to TOKYO_DS.pubs2
set error class tokyo_error_class
set function string class tokyo_func_class
```

```
set username pubs2_maint
set password pubs2_maint_pw
```

This command assigns the `tokyo_func_class` function-string class to the database connection.

Example for altering an existing connection

The following command alters an existing database connection to specify a different function-string class:

```
alter connection to TOKYO_DS.pubs'2
set function string class tokyo_func_class2
```

Dropping a function-string class

If you are sure that you will not need it again, you may want to drop a function-string class that you created from the replication system. You can drop any function-string class except the three system-provided classes and any user-created class that currently serves as a parent class. Before you can drop a function-string class, you must drop all database connections that use the function-string class, or you can alter the connections to use a different class.

Dropping a function-string class deletes all function strings defined for the class and removes all references to the class from the RSSD.

RSM

For instructions on dropping a function-string class in Sybase Central, see “Deleting function-string classes” in Replication Server’s plug-in help.

To drop a function-string class from the `isql` command line, use the `drop function string class` command. For example, the following command drops the `tokyo_func_class` function-string class and all of its function strings:

```
drop function string class tokyo_func_class
```

Enter this command in the Replication Server that is the primary site for the class.

Refer to “drop function string class” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information.

Managing function strings

Each destination Replication Server uses function strings to convert the functions to commands that are appropriate for the destination data server (such as Adaptive Server) before it submits these commands. See Chapter 2, “Replication Server Technical Overview” for more information about DSI threads, the components that perform this conversion at the replicate Replication Server.

The following sections describe elements of function strings and the commands for managing them. Refer to the *Replication Server Reference Manual* for complete command syntax and permissions.

RSM

For instructions on managing function strings in Sybase Central, see “Function Strings” in Replication Server’s plug-in help.

Function strings and function-string classes

If you do not require customized function strings, you can use one of the system-provided function-string classes to provide default function strings. If you require customized strings, you must use the system-provided class—`rs_sqlserver_function_class`—in which you can customize function strings or create a derived or base function-string class. See “Function-string classes” on page 445 for details.

- If the connection for the database in which the function will be executed uses a system-provided function-string class or a derived class that inherits directly or indirectly from `rs_default_function_class` or a function-string class for a non-Sybase data server, default function strings are provided for every system function and user-defined function.
- If the connection uses a user-created base function-string class (which does not inherit function strings) or a derived class that inherits from such a class, you must create function strings for every system function and user-defined function. Create them in the base class if you want them to be available in all its derived classes.

Function-string input and output templates

To customize function strings, you alter their input and/or output templates. Depending on the function, function strings may include both an input template and an output template, an output template, or neither template:

- For the `rs_select` and `rs_select_with_lock` functions, used in subscription materialization, Replication Server uses input templates to locate the function string that corresponds to a subscription's `where` clause.
- For all functions Replication Server uses output templates to map functions to the language commands or to apply RPC invocations at the destination data server.

Requirements for using input and output templates

When you alter templates to customize function strings, you should keep in mind the following requirements:

- Function-string input and output templates are limited to 64K bytes. The result of substituting runtime values for embedded variables in function-string input or output templates must not exceed 64K.
- Function-string input and output templates are delimited with single quotation marks (`'`).
- Function-string variables are enclosed within a pair of question marks (`?`).
- A variable name and its modifier are separated with an exclamation mark (`!`).

Language output templates involve additional related requirements. See “Using output templates” on page 458 for details.

Using output templates

You alter output templates to customize function strings. Replication Server uses output templates to determine the format of the command sent to a data server. Most output templates use one of two formats: language or RPC, corresponding to the format of the function string itself. (See “Function strings” on page 442 for information on function-string formats.) An output template for an `rs_writetext` function string can use the RPC format or one of the additional formats `writetext` or `none`, but not a language output template. See “Using function strings with text, image, and rawobject datatypes” on page 475 for details.

Language output templates

Language output templates contain text that the data server interprets as commands. Replication Server substitutes values for variables embedded in the output template and passes the resulting language command(s) to the data server to process.

See “Creating function strings” on page 464 for example output templates. See “Using function-string variables” on page 462 for details on embedded variables.

Within a language output template, Replication Server interprets certain characters in special ways:

- Two single quote characters (") are interpreted as one single quote
- Two question marks (??) are interpreted as one single question mark
- Two semicolons (;;) are interpreted as one single semicolon

Other than the embedded variable substitutions and these special interpretations, Replication Server does not attempt to interpret the contents of language output templates.

See “Function-string variable formatting” on page 463 for information about how Replication Server formats function-string variables when it maps function strings to data server commands.

RPC output templates

Unlike language output templates, Replication Server interprets the contents of RPC output templates. They are written in the format of the Transact-SQL `execute` command. Replication Server parses the output template to construct a remote procedure call to send to the Adaptive Server, Open Server gateway, or Open Server application.

RPC output templates work well with gateways or Open Servers with no language parser. RPCs are usually more compact than language requests and, since they do not require parsing by the data server, may also be more efficient. Therefore, you might choose to use an RPC even when a data server supports language requests.

Output templates for *rs_writetext* function strings

Replication Server supports three output formats for creating an `rs_writetext` function string: `RPC`, `writetext`, and `none`. The `writetext` and `none` output templates can only be used in `rs_writetext` function strings.

See “Using function strings with text, image, and rawobject datatypes” on page 475 for more information about writetext and none.

Using input templates

Input templates are used only for non-bulk materialization and for dematerialization with purge—those situations where Replication Server must select data to add or delete from selected tables. `rs_select` and `rs_select_with_lock` are the only function strings that can contain input templates. Replication Server determines which function string to use with a subscription during materialization or dematerialization by:

- Matching the subscription’s replication definition
- Matching the input template with the where clause used in the subscription

`rs_select` and `rs_select_with_lock` also contain output templates to specify the actual select statements or other operations that perform the desired materialization or dematerialization.

For the system-provided classes, Replication Server generates default function strings for the `rs_select` and `rs_select_with_lock` functions when you create a replication definition. Generally, you only need to customize these function strings if multiple subscriptions exist for your replication definition.

Function strings for the `rs_select` and `rs_select_with_lock` functions are most often used for materialization. If you plan multiple subscriptions to the same replication definition, create the function strings before you create the subscriptions. See “Subscription materialization methods” on page 351 for more information about subscription materialization.

Function strings for `rs_select` and `rs_select_with_lock` may also be used for subscription dematerialization, which uses the where clause of the command used to create the subscription. The function strings for these functions must exist before you drop the subscriptions. See “Using the drop subscription command” on page 378 for more information about dematerialization.

An input template can contain user-defined variables whose values come from constants in the where clause of a subscription. No other types of function-string variables are allowed in input templates. An output template in the same function string can reference these user-defined variables.

If you need to customize an output template to select materialization data, you can omit the input template from an `rs_select` or `rs_select_with_lock` function string. Doing so creates a default function string that can match any select statement when no other function string's input template matches the select command.

As with other functions with replication-definition scope, you create function strings for the `rs_select` and `rs_select_with_lock` functions in the primary Replication Server where the replication definition was created.

Class in which to create function strings

When you create `rs_select` and `rs_select_with_lock` function strings for materialization, you create them in the function-string class that is assigned to the connection to the primary database from which you are selecting materialization data. If you are using bulk materialization, you do not need to create `rs_select` and `rs_select_with_lock` function strings for materialization.

When you create `rs_select` and `rs_select_with_lock` function strings for dematerialization, you create them in the function-string class that is assigned to the connection to the replicate database for which you are selecting data to be dematerialized. If you drop a subscription using `drop subscription` with the `without purge` option, you do not need `rs_select` and `rs_select_with_lock` function strings for dematerialization.

Example for `rs_select` function string

In the following example, a site subscribes to a specified publisher's book titles through the replication definition `titles_rep`. There must be an `rs_select` function string with an input template that compares the publisher column in the `pubs2` database's `titles` table to a user-defined value that identifies the publisher.

The `create function string` command creates a function string with an input template that compares the publisher column `pub_id` to the user-defined variable `?pub_id!user?`. For details on function-string variables, see "Using function-string variables" on page 462.

The input template matches any subscription with a `where` clause of the form `where pub_id = constant`. As a result, the output template, when it is used, includes the `constant` value. The output template selects materialization data from two different tables.

```
create function string titles_rep.rs_select;pub_id
  for sqlserver2_function_class
scan 'select * from titles where pub_id =
      ?pub_id!user?'
output language
      'select * from titles where pub_id =
```

```

        ?pub_id!user?
union
select * from titles.pending where pub_id =
        ?pub_id!user?'
```

See “Creating function strings” on page 464 for details. Refer to the *Replication Server Reference Manual* for complete syntax.

Using function-string variables

Variables embedded in function-string input or output templates are symbolic markers for various runtime values.

A variable can represent a column name, the name of a system-defined variable, the name of a parameter in a user-defined function, or a user-defined variable defined in an input template. The variable must refer to a value with the same datatype as anything to which it is assigned.

Function-string variables are enclosed inside of a pair of question marks (?), as shown:

```
?variable!modifier?
```

The *modifier* portion of a variable identifies the type of data the variable represents. The modifier is separated from the variable name with an exclamation point (!).

Replication Server recognizes the modifiers listed in Table 14-3:

Table 14-3: Function-string variable modifiers

Modifier	Description
new, new_raw	A reference to the new value of a column in a row that Replication Server is inserting or updating.
old, old_raw	A reference to the <i>old</i> values of a column in a row that Replication Server is inserting or updating.
user, user_raw	A reference to a variable that is defined in the input template of an <code>rs_select</code> or <code>rs_select_with_lock</code> function string.
sys, sys_raw	A reference to a system-defined variable.
param, param_raw	A reference to a stored-procedure parameter

Modifier	Description
text_status	<p>A reference to the text_status value for text or image data. Possible values are:</p> <ul style="list-style-type: none"> • 0x000 – Text field contains NULL value, and the text pointer has not been initialized. • 0x0002 – Text pointer is initialized. • 0x0004 – Real text data will follow. • 0x0008 – No text data will follow because the text data is not replicated. • 0x0010 – The text data is not replicated but it contains NULL values.

Note Function strings for user-defined functions may not use the *new* or *old* modifiers.

Refer to “create function string” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for a list of system-defined variables that you can use in function-string input or output templates.

See “Using the default system variable” on page 473 for information on applications for that system variable.

Function-string variable formatting

When Replication Server maps function-string output templates to data server commands, it formats the variables using the Adaptive Server format.

For most variables (except those special cases with modifiers ending in *_raw*), Replication Server formats data as follows:

- Adds an extra single-quote character to single-quote characters appearing in character and date/time values.
- Adds single-quote characters around character and date/time values, if they are missing.
- Adds the appropriate monetary symbol (for example, the dollar sign) to values of money datatypes.
- Adds the “0x” prefix to values of binary datatypes.
- Adds a combination of a backslash (\) and newline character between existing instances of a backslash and newline character in character values. Adaptive Server treats a backslash followed by a newline as a continuation character and, therefore, deletes the added pair of characters, leaving the original characters intact.

Replication Server does not alter datatypes in these ways for modifiers that end in `_raw`.

Creating function strings

To add a function string to a function-string class, use the create function string command. Enter function-string commands at the primary site of the function string:

- For function strings with replication-definition scope, the primary site is the Replication Server where the replication definition was created.
- For function strings with class scope, the primary site is the Replication Server that is the primary site for the class. The primary site for a derived class is the same as for its parent class, unless the parent class is one of the system-provided classes. See “Primary site for a function-string class” on page 453 for more information.

If you are using a derived function-string class whose parent class is not provided by the system, you may choose to customize function strings in the parent class rather than in the derived class that is actually assigned to a particular database connection. Doing so would make the customized function strings available for any additional derived classes of that parent class.

Guidelines for creating function strings

The following guidelines for creating function strings pertain to function-string classes:

- If you need to customize function strings, you can do so in any class other than the system-provided classes `rs_default_function_class` and `rs_db2_function_class`.
- You must assign a function-string class a primary site before you can create function strings for the class. The system-provided class `rs_sqlserver_function_class` has no primary site until you assign one using the create function string class command.
- If the function-string class is a new base class, you must create function strings for all the necessary system functions before you can use the class.

The following guidelines pertain to function strings themselves:

- You can specify an optional name for the function string. For the `rs_select`, `rs_select_with_lock`, `rs_datarow_for_writetext`, `rs_get_textptr`, `rs_textptr_init`, and `rs_writetext` functions, Replication Server uses the function-string name to uniquely identify the function strings. Function string names are unique when you qualify them fully.
- If the input template is omitted for an `rs_select` or `rs_select_with_lock` function string, Replication Server matches any subscriptions that do not have matching function strings.
- If you are customizing function strings for functions with replication-definition scope, you must create the function strings before you create the subscriptions.
- You can put several commands in a language output template, separating them with semicolons. See “Defining multiple commands in a function string” on page 470 for details.

Make sure that the database connection batch parameter has been set to allow command batching. See “Configuration parameters affecting individual connections” on page 180.

- You can use Adaptive Server syntax to specify a null value for a *constant* in a function string.

Refer to “create function string” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for the complete syntax for the create function string command.

Example for `rs_begin`
function string

The following example creates a function string for the `rs_begin` function that begins a transaction in the database by executing a stored procedure named `begin_xact`.

```
create function string rs_begin
  for gateway_func_class
  output rpc 'execute begin_xact'
```

Example for `rs_insert`
function string

The following example creates a function string for a `rs_insert` function that references the `publishers_rep` replication definition, which executes an RPC at the replicate database as a result of an insert in the primary table. The stored procedure `insert_publisher` is defined only at the replicate database.

```
create function string publishers_rep.rs_insert
  for rs_sqlserver_function_class
  output rpc
  'execute insert_publisher
  @pub_id = ?pub_id!new?,
  @pub_name = ?pub_name!new?,'
```

```
@city = ?city!new?,  
@state = ?state!new?'
```

Altering function strings

The alter function string command replaces an existing function string. alter function string acts essentially the same as create function string except that it executes the drop function string command first. The function string is dropped and re-created in a single transaction to prevent any errors from occurring as a result of missing function strings.

You can alter a function string using either the alter function string command or the create function string command. To alter a function string using the create function string command, you must include the optional clause with overwrite after the name of the function-string class. This command drops and re-creates an existing function string, the same as the alter function string command.

To alter a function string using the alter function string command, you must first create a function string.

In a derived class, first use the create function string command to override the function string that is inherited from the parent class. You cannot alter a function string in a derived class unless the function string has been explicitly created for the derived class.

You alter function strings at the Replication Server that is the primary site for the existing function string:

- For functions of replication-definition scope, alter the function string at the primary Replication Server where the replication definition was defined.
- For functions of class scope, alter the function string at the primary site for the function-string class. The primary site for a derived class is the same as for its parent class, unless the parent class is one of the system-provided classes. See “Primary site for a function-string class” on page 453 for more information.

For system functions that allow multiple function-string mappings, such as rs_select and rs_select_with_lock, provide the complete function string name in the alter function string syntax. Replication Server uses the name to determine which function string to alter.

See “Creating function strings” on page 464 for example function strings.

Refer to “alter function string” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for the complete syntax for the alter function string command.

RSM

For instructions on altering function strings in Sybase Central, see “Changing function strings” in Replication Server’s plug-in help.

Dropping function strings

To discard a customized function string in a derived class and restore the function string from the parent class, drop the function string. Use the drop function string command to remove one or more function strings in a function-string class.

Warning! If you want to drop and re-create a function string, use alter function string to replace an existing function string with a new one. Dropping and then re-creating a function string by other methods can lead to a state where the function string is temporarily missing.

If a transaction that uses this function string occurs between the time the function string is dropped and the time it is re-created, Replication Server detects the function string as missing and fails the transaction.

When you drop the function string from a derived class, you restore the function string from the parent class.

RSM

For instructions on dropping function strings in Sybase Central, see “Deleting function strings” in Replication Server’s plug-in help.

Refer to “drop function string” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information on this command.

You can also drop customized function strings from the system-provided class `rs_sqlserver_function_class`.

To restore a default function string for a function string with replication-definition scope that you have dropped, use the alter function string command to omit the output clause. See “Restoring default function strings” on page 469 for details.

Examples

The following command drops the `rs_insert` function string for the `publishers_rep` replication definition in the class `sqlserver2_func_class`:

```
drop function string
```

```
publishers_rep.rs_insert  
for sqlserver2_func_class
```

The following command drops the `pub_id` instance of a function string for the `rs_select` function for the `publishers_rep` replication definition in the class `derived_class`. Drop function strings for the `rs_select_with_lock` function in a similar way.

```
drop function string  
publishers_rep.rs_select;pub_id  
for derived_class
```

The following command drops the `rs_begin` function string from the `gateway_func_class` function-string class:

```
drop function string rs_begin  
for gateway_func_class
```

Dropping all function strings for a function

In cases where there are multiple function strings for a specified function, you can drop all function strings for that function simultaneously.

The following command drops all function strings for the `rs_select_with_lock` function that references the `publishers_rep` replication definition in the class `sqlserver2_func_class`:

```
drop function string  
publishers_rep.rs_select_with_lock;all  
for sqlserver2_func_class
```

System functions that can have multiple function string mappings include the `rs_select`, `rs_select_with_lock`, `rs_get_textptr`, `rs_textptr_init`, or `rs_writetext` functions.

Examples of using the *all* keyword as shorthand

When dropping function strings for any system function for which you provided a lengthy name, you can use the `all` keyword as shorthand for the name of the function string instance. For example, the following command gives a long name for a function string:

```
create function string  
publishers_rep.rs_insert;my_insert_function_string  
for sqlserver2_func_class  
...
```

In this case, the following command drops the function string without you having to enter the fully qualified name:

```
drop function string  
publishers_rep.rs_insert;all  
for sqlserver2_func_class
```

Restoring default function strings

To restore the Adaptive Server default function string for a system function with replication definition scope, omit the output clause in the create function string or alter function string command. You cannot omit an output template from a system function with function-string-class scope, although you can specify an empty template.

RSM

For instructions on restoring default function strings in Sybase Central, see “Reverting to original function strings” in Replication Server’s plug-in help.

In all classes, even derived classes, executing the create function string or alter function string command without the output clause restores the same function string that is provided by default for the system-provided classes `rs_sqlserver_function_class` and `rs_default_function_class`.

The default function-string definition this method yields may or may not be appropriate for the databases to which you have assigned the class. This method may be most helpful when you are using a customized `rs_sqlserver_function_class` or when you are using other user-created base classes for Adaptive Server databases.

In a derived class, if you want to discard a customized function string and restore the function string from the parent class, drop the function string. See “Dropping function strings” on page 467 for details.

Example for *alter function string*

The following command replaces a customized `rs_insert` function string for the `publishers_rep` replication definition with the default function string:

```
alter function string publishers_rep.rs_insert
for rs_sqlserver_function_class
```

See “Altering function strings” on page 466 for details on using the alter function string command.

Example for *create function string* in a derived class

You can use this method in a derived function-string class to override an inherited function string with the Adaptive Server default function string. The following command replaces an inherited `rs_insert` function string for the `publishers_rep` replication definition with the default function string:

```
create function string publishers_rep.rs_insert
for derived_class
```

See “Creating function strings” on page 464 for details on using the create function string command.

Creating empty function strings with the output template

You can create an empty function string—one that performs no action—by including the output language clause with an empty function string specified with two single quotes.

For example, the following command defines no action for the `rs_insert` function string for the `publishers_rep` replication definition:

```
alter function string publishers_rep.rs_insert
for derived_class
output language ''
```

See “Altering function strings” on page 466 for details on using the `alter function string` command.

Remapping table and column names with function strings

You can use function strings to translate the table name and column names for a replicated table to names other than those specified in the replication definition. The function strings that Replication Server generates for the `rs_sqlserver_function_class` function-string class use the names specified by the replication definition for the table, but you can define your own function strings with any names you like.

This procedure is useful if a site has existing client applications that use different table and column names than those defined by the replication definition for the primary data. Customizing function strings allows Replication Server to maintain the data in the table and does not require that you alter the site’s applications.

To do this, you can use either language function strings or RPC function strings with Adaptive Server stored procedures at the remote site.

Defining multiple commands in a function string

Language output templates can contain many commands. Adaptive Server permits multiple commands in a batch. Although most other data servers do not offer this feature, Replication Server allows you to batch commands in function strings for any data server by separating commands with a semicolon (;).

Use two consecutive semicolons (;;) to represent a semicolon that is *not* to be interpreted as a command separator.

If the data server supports command batches, Replication Server replaces the semicolons with the DSI command separator character (`dsi_cmd_separator` configuration parameter), as necessary, and submits the commands in a single batch.

If the data server does not support command batches, Replication Server submits each command in the function string separately.

For example, the output template in the following function string contains two commands:

```
create function string rs_commit
for sqlserver2_function_class
output language
'execute rs_update_lastcommit
  @origin = ?rs_origin!sys?,
  @origin_qid = ?rs_origin_qid!sys?,
  @secondary_qid = ?rs_secondary_qid!sys?;
commit transaction'
```

Support for batches is enabled or disabled in Replication Server with the `alter connection` command.

Set `batch` to “on” to allow command batching for a database, or set it to “off” to send individual commands to the data server. The default is “on.”

To set batching “on” for this example, enter:

```
alter connection to SYDNEY_DS.pubs2
set batch to 'on'
```

To set batching “off,” enter:

```
alter connection to SYDNEY_DS.pubs2
set batch to 'off'
```

Using declare statements in language output templates

To include declare statements, used to define local variables, in the language output templates, make sure that the batch configuration parameter is set to “off” for the Replication Server connected to the database. When batch is set to “on” (the default), Replication Server can send multiple invocations of a function string to the data server as a single command batch, thereby putting multiple declarations of the same variable in that batch, which is unacceptable to Adaptive Server.

Performance is slower when batch mode is off because Replication Server must wait for a response to each command before the next one is sent. If your performance requirements are low, you can use declare statements in your function strings if you set batch to “off.” Alternatively, if you want to use batch mode for improved performance, create function-string language output templates that execute stored procedures, which can include declare statements and other commands.

Refer to “Setting and changing parameters affecting physical connections” on page 178 for more information about batch.

Displaying function-related information

You can obtain information about existing function strings and classes in your replication system in two ways:

- Using Replication Server admin command
- Using Adaptive Server stored procedures

RSM

For instructions on viewing information about function strings and classes in Sybase Central, see “Viewing function string properties” in Replication Server’s plug-in help.

Obtaining information using the *admin* command

You can display the names of the function-string classes used in your Replication Server system using one of Replication Server’s admin commands.

Use `admin show_function_classes` to display the names of existing function-string classes and their parent classes. It also indicates the inheritance level of the class. Level 0 is a base class such as `rs_default_function_class` or `rs_db2_function_class`, level 1 is a derived class that inherits from a base class, and so on.

For example:

```

                                admin show_function_classes
Class                            ParentClass                      Level
-----                            -
sql_derived_class                rs_default_function_class        1
rs_db2_derived_class             rs_db2_function_class            1
    
```

```
rs_db2_function_class
...
```

0

For more information about this command, see Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual*.

Obtaining information using stored procedures

You can obtain information about existing functions, function strings, and function-string classes in your system using stored procedures in a Replication Server’s RSSD.

Refer to Chapter 6, “Adaptive Server Stored Procedures,” in the *Replication Server Reference Manual* for more information about these stored procedures.

rs_helpfunc

rs_helpfunc displays information about system functions and user-defined functions for a Replication Server or for a particular table or function replication definition. The syntax is:

```
rs_helpfunc [replication_definition [, function_name]]
```

rs_helpstring

rs_helpstring displays the parameters and function-string text for functions associated with a replication definition. The syntax is:

```
rs_helpstring replication_definition
[, function_name]
```

rs_helpclass

rs_helpclass lists all function-string classes and error classes and their primary Replication Servers. The syntax is:

```
rs_helpclass [class_name]
```

rs_helpclassstring

rs_helpclassstring displays the function-string information for class-scope functions. The syntax is:

```
rs_helpclassstring class_name [, function_name]
```

Using the default system variable

The *rs_default_fs* system variable allows you to perform the following tasks:

- Extend function strings with replication-definition scope to include additional commands (such as those for auditing or tracking)

- Customize `rs_update` and `rs_delete` function strings and still be able to use the `replicate minimal columns` option in your replication definitions

Note Function strings containing the `rs_default_fs` system variable may only be applied on Adaptive Servers or data servers that accept Adaptive Server syntax. Otherwise, errors will occur.

Refer to “create function string” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for a complete list of function string system variables.

Extending default function strings

You can use the `rs_default_fs` system variable with all function strings that have replication-definition scope (table or function) as a way to extend the default function-string behavior.

Using the `rs_default_fs` system variable reduces the amount of typing required when you want to keep the functionality of the default function string intact and include additional commands. For example, you can add commands to extend the capabilities of the default function string for auditing or tracking purposes.

Commands that you add to the output language template may either precede or follow the `rs_default_fs` system variable. They may or may not affect how the row is replicated into the replicate table.

The following example shows how you might use the `rs_default_fs` system variable in the create function string command (or the alter function string command) to verify that an update has occurred:

```
create function string replication_definition.rs_update
  for function_string_class
  output language '?rs_default_fs!sys?;

if (@@rowcount = 0)
  begin
    raiserror 99999 "No rows updated!"
  end'
```

In this example, the `rs_default_fs` system variable, embedded in the language output template, maintains the functionality of the default `rs_update` function string while the output template then checks to see if any rows have been updated. If they have not been updated, Replication Server raises an error.

In this example, the commands that follow the system variable do not affect how the row is to be replicated at the replicate site. You can use the *rs_default_fs* system variable with similar additional commands for verification or auditing purposes.

Using *replicate minimal columns*

If you have specified replicate minimal columns for a replication definition, you normally cannot create non-default function strings for the *rs_update*, *rs_delete*, *rs_get_textptr*, *rs_textptr_init*, or *rs_datarow_for_writetext* system functions.

You can create non-default function strings for the *rs_update* and *rs_delete* functions by embedding the *rs_default_fs* system variable in the output language template of the create function string or alter function string commands and still use the minimal columns option.

You cannot use any variables, including the *rs_default_fs* system variable, that access non-key column values in *rs_update* or *rs_delete* function strings for replication definitions that use the minimal columns option. When you create such a function string, you may not know ahead of time which columns will be modified at the primary table. You may, however, include variables that access key column values.

See “create replication definition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information about the replicate minimal columns option.

Using function strings with *text*, *image*, and *rawobject* datatypes

In an environment that supports text, image, and rawobject datatypes, you can customize function strings for the *rs_writetext* function using the output template formats *writetext* or *none*. The methods discussed in this section can only be used with *rs_writetext* function strings.

RSM

For instructions on using text, image, and rawobject datatypes in function strings in Sybase Central, see “Customizing function strings for text and image columns” in Replication Server’s plug-in help.

For Replication Server version 11.5 or later, you can use multiple replication definitions instead of function strings. See Chapter 9, “Managing Replicated Tables” for information about multiple replication definitions.

Using *output writetext* for *rs_writetext* function strings

The *writetext* output template option for *rs_writetext* function strings instructs Replication Server to use the Client-Library function *ct_send_data* to update a text, image, or rawobject column value. It specifies logging behavior for text, image, and rawobject columns in the replicate database.

writetext output templates support the following options:

- *use primary log* – logs the data in the replicate database, if the logging option was specified in the primary database.
- *with log* – logs the data in the replicate database transaction log.
- *no log* – does not log the data in the replicate database transaction log.

Using *output none* for *rs_writetext* function strings

The *none* output template option for *rs_writetext* function strings instructs Replication Server not to replicate a text or image column value. This option provides necessary flexibility for using text and image columns within a heterogeneous environment.

Heterogeneous replication and *text*, *image*, and *rawobject* data

To replicate text, image, and rawobject data from a foreign data server into an Adaptive Server database, you must include the text, image, and rawobject data in the replication definition so that a subscription can be created for the Adaptive Server database. However, you might not want to replicate the text, image, and rawobject data into other replicate data servers, whether they are other foreign data servers or other Adaptive Servers.

With the *none* output template option, you can customize *rs_writetext* function strings to map operations to a smaller table at a replicate site and to instruct the *rs_writetext* function string not to perform any text, image, or rawobject operation against the replicate site.

There is one `rs_writetext` function string for each text, image, and rawobject column in the replication definition. If you do not want to replicate a certain text, image, or rawobject column, customize the `rs_writetext` function string for that column. Specify the column name in the create or alter function string command, as shown in the example below. You may also need to customize the `rs_insert` function string.

Example

Assume that a replication definition does not allow null values in a text, image, or rawobject column and that you do not require certain text, image, or rawobject columns at the replicate site.

If inserts occur in those columns at the primary site, you must customize the `rs_writetext` function strings for the text, image, or rawobject columns that are not needed at the replicate site. You must also customize the `rs_insert` function string for the replication definition.

For example, assume that you have primary table `foo`:

```
foo (int a, b text not null, c image not null)
```

In `foo`, you perform the following insert:

```
insert foo values (1, "111111", 0x11111111)
```

By default, Replication Server translates `rs_insert` into the following form for application by the DSI thread into the replicate table `foo`:

```
insert foo (a, b, c) values (1, "", "")
```

The DSI thread calls:

- `ct_send_data` to insert text data into column `b`
- `ct_send_data` to insert image data into column `c`

Because null values are not allowed for the text column `b` and the image column `c`, the DSI thread shuts down if the replicate table does not contain either column `b` or column `c`.

If the replicate table only contains columns `a` and `b`, you need to customize the `rs_writetext` function for column `c` to use output `none`, as follows:

```
alter function string foo_repdef.rs_writetext;c
for rs_sqlserver_function_class
output none
```

You must specify the column name (`c` in this example) as shown to alter the `rs_writetext` function string for that column.

If the replicate table only contains columns a and b, you also need to customize the `rs_insert` function string for the replication definition so that it will not attempt to insert into column c, as follows:

```
alter function string foo_repdef.rs_insert
  for rs_sqlserver_function_class
  output language
  'insert foo (a, b) values (?a!new?, '')'
```

You do not have to customize `rs_insert` if the replication definition specifies that null values are allowed for column c. By default, `rs_insert` does not affect any text or image columns where null values are allowed.

Managing Warm Standby Applications

This chapter describes one way to create and manage a warm standby application using Replication Server.

Name	Page
Overview	480
What information is replicated?	485
Setting up warm standby databases	494
Switching the active and standby databases	506
Monitoring a warm standby application	514
Setting up clients to work with the active data server	516
Altering warm standby database connections	518
Warm standby applications using replication	524
Using replication definitions and subscriptions	531
Loss detection and recovery	542

This chapter describes how to set up and configure a warm standby application between two Adaptive Server databases—the primary or active database and a single standby database. Changes to the primary database are copied directly to the warm standby database. To change or qualify the data sent, you must add table and function replication definitions.

You can also use multi-site availability (MSA) to set up a warm standby application between Adaptive Server databases. MSA enables replication to multiple standby and replicate databases. You can choose whether to replicate the entire database or replicate (or not replicate) specified tables, transactions, functions, system stored procedures, and data definition language (DDL). See Chapter 12, “Managing Replicated Objects Using Multi-Site Availability,” for information about setting up a warm standby application using MSA.

Overview

A **warm standby application** is a pair of Adaptive Server or SQL Server databases, one of which is a backup copy of the other. Client applications update the **active database**; Replication Server maintains the **standby database** as a copy of the active database.

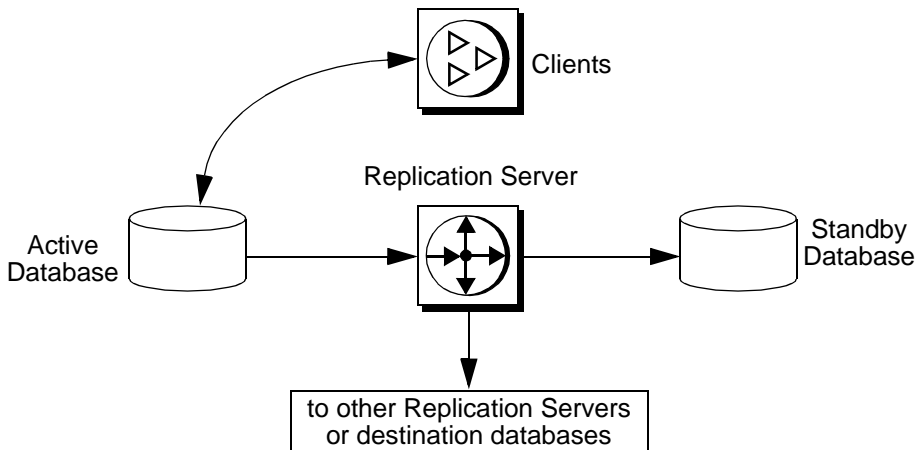
If the active database fails, or if you need to perform maintenance on the active database or on the data server, a switch to the standby database allows client applications to resume work with little interruption.

To keep the standby database consistent with the active database, Replication Server reproduces transaction information retrieved from the active database's transaction log. Although replication definitions facilitate replication into the standby database, they are not required. Subscriptions are not needed to replicate data into the standby database.

How a warm standby works

Figure 15-1 illustrates the normal operation of an example warm standby application.

Figure 15-1: Warm standby application



In this warm standby application:

- Client applications execute transactions in the active database.

- The RepAgent for the active database retrieves transactions from the transaction log and forwards them to Replication Server.
- Replication Server executes the transactions in the standby database.
- Replication Server may also copy transactions to destination databases and remote Replication Servers.

See Figure 15-4 on page 507 for more details about the components and processes in a warm standby application.

Database connections in a warm standby application

In a warm standby application, the active database and the standby database appear in the replication system as a connection from the Replication Server to a single logical database. The Replication System Administrator creates this **logical connection** to establish one symbolic name for both the active and standby databases.

Thus, a warm standby application involves three database connections from the Replication Server:

- A physical connection for the active database
- A physical connection for the standby database
- A logical connection for the active and standby databases

Replication Server maps the logical connection to the currently active database and copies transactions from the active to the standby database.

See “Setting up warm standby databases” on page 494 for details on creating the logical and physical database connections. See Chapter 7, “Managing Database Connections” for more information about physical database connections.

Primary and replicate databases and warm standby applications

In many Replication Server applications:

- A primary database is the source of data that is copied to other databases through the use of replication definitions and subscriptions.
- A destination database receives data from the primary (source) database.

Replication Server treats a logical database like any other database. Depending on your application, the logical database in a warm standby application may function as:

- A primary database, or
- A replicate database, or
- A database that does not participate in replication

See “Switching the active and standby databases” on page 506 for more information about warm standby applications that do not participate in standard replication.

See “Warm standby applications using replication” on page 524 for more information about warm standby applications for primary or replicate databases.

Comparison of database relationships

In most of this book, databases are defined as “primary” or “replicate.” In discussing warm standby applications, however, databases are also defined as “active” or “standby.” Table 15-1 explains the difference.

Table 15-1: Active and standby vs. primary and destination databases

Active and standby databases	Primary and replicate databases
The active and standby databases must be managed by the same Replication Server.	Primary and destination databases may be managed by the same or different Replication Servers.
The active and standby databases must be Adaptive Server (or SQL Server) databases.	Except where they participate in warm standby applications, primary and destination databases need not be Adaptive Server (or SQL Server) databases.
The active database has one standby database. Information is always copied from the active to the standby database.	A primary database can have one or more destination databases. Some databases contain both primary and copied data.
The use of replication definitions is optional. Subscriptions are not used.	Replication definitions and subscriptions are required for replication from a primary to a destination database.
The connection to the standby database uses the function-string class <code>rs_default_function_class</code> . You cannot customize function strings for this class.	The connection to a replicate database can use a function-string class in which you can customize function strings. For example, it may use a derived class that inherits function strings from <code>rs_default_function_class</code> .
You can switch the roles of the active and standby databases.	You cannot switch the roles of primary and replicate databases.

Active and standby databases	Primary and replicate databases
<p>Client applications generally connect to the active database. (However, you can perform read-only operations at the standby database.)</p> <p>No mechanism is provided for switching client applications when you switch the Replication Server to the standby database.</p>	<p>Client applications can connect to either primary or destination database. Only primary data can be directly modified.</p> <p>Generally, client applications do not need to switch between primary and destination databases.</p>
<p>The RepAgent for the active database submits all transactions on replicated tables, including maintenance user transactions, to the Replication Server, which reproduces them in the standby database.</p> <p>In a warm standby application for a destination database, transactions in the active database are normally executed by the maintenance user.</p>	<p>In most applications, RepAgent does not submit maintenance user transactions to the Replication Server to be reproduced in destination databases.</p> <p>The maintenance user does not generally execute transactions in primary databases.</p>

Warm standby requirements and restrictions

The following restrictions apply to all Replication Server warm standby applications:

- You must use a Sybase Adaptive Server or SQL Server that supports warm standby applications. Refer to your release bulletin for more information.
- One Replication Server manages both the active and standby databases. Both the active and standby databases must be Adaptive Server or SQL Server databases.
- You cannot create a standby database for the RSSD or for the master database.
- Replication Server does not switch client applications to the standby database. See “Setting up clients to work with the active data server” on page 516 for more information.
- You should run Adaptive Server for the active and standby databases on different machines. Putting the active and standby databases on the same data server or hardware resources undermines the benefits of the warm standby feature.
- Although Adaptive Server allows tables that contain duplicate rows, tables in the active and standby databases should have unique values for the primary key columns in each row.

- Failover support is not a substitute for warm standby. While warm standby keeps a copy of a database, Failover support accesses the same database from a different machine. Failover support works the same for connections from Replication Server to warm standby databases.

For more detailed information about how Sybase Failover works in Adaptive Server, refer to *Using Sybase Failover in a High Availability System*, which is part of the Adaptive Server Enterprise version 12.0 documentation set.

For more detailed information about how Failover support works in Replication Server, see “Configuring the Replication System to Support Sybase Failover” in Chapter 16, “Replication System Recovery,” of the *Replication Server Administration Guide*.

- The commands and procedures for abstract plans are replicated, except for the following:
 - The and set `@plan_id` clause of create plan is not replicated. For example, this command is not replicated as shown.

```
create plan "select avg(price)
from titles" "(t_scan titles)
into dev_plans and set @plan_id
```

Rather, it is replicated as:

```
create plan "select avg(price)
from titles" "(t_scan titles)
into dev_plans
```

- The abstract plan procedures that take a plan ID as an argument (`sp_drop_qplan`, `sp_copy_qplan`, `sp_set_qplan`) are not replicated.
- The set plan command is not replicated.

Function strings for maintaining standby databases

Replication Server uses the system-provided function-string class `rs_default_function_class` for the standby DSI, which is the connection to the standby database. Replication Server generates default function strings for this class. You cannot customize the function strings in the class `rs_default_function_class`.

What information is replicated?

Replication Server supports different methods for enabling replication to the standby database. The level and type of information that Replication Server copies to the standby database depends on the method you choose.

You must choose one of these two methods:

- Use the `sp_reptostandby` system procedure to mark the entire database for replication to the standby database. `sp_reptostandby` enables replication of data manipulation language (DML) commands and a set of supported data definition language (DDL) commands and system procedures.
 - DML commands, such as insert, update, delete, and truncate table, change the data in user tables.
 - DDL commands and system procedures change the schema or structure of the database.

`sp_reptostandby` allows replication of DDL commands and procedures that make changes to system tables stored in the database. You can use DDL commands to create, alter, and drop database objects such as tables and views. Supported DDL system procedures affect information about database objects. They are executed at the standby database by the original user.

- If you use SQL Server version 11.0.x, or you choose not use `sp_reptostandby`, you can mark individual user tables for replication with `sp_setreptable`. This procedure enables replication of DML operations for the marked tables.

Optionally, you can also tell Replication Server which user stored procedures to replicate to the standby database:

- If you use Adaptive Server or SQL Server version 11.0.x, you can copy the execution of user stored procedures to the standby database by marking them with the `sp_setrepproc` system procedure. Normally, only stored procedures associated with function replication definitions are replicated to standby databases.

Refer to “Using `sp_setrepproc` to copy user stored procedures” on page 491 for more information.

Comparing replication methods

Table 15-2 compares `sp_reptostandby` and `sp_setreptable`, detailing how each copies information to the standby database. Many of these issues are discussed in detail later in the chapter.

Table 15-2: Comparison of table replication methods

<code>sp_reptostandby</code>	<code>sp_setreptable</code>
Copies all user tables to the standby database.	Lets you choose which user tables are copied to the standby database.
Allows replication of DML commands and supported DDL commands and system procedures. Supported DDL operations are listed in “Supported DDL commands and system procedures” on page 488.	Allows replication of DML commands executed on marked tables. Note Supported DDL operations can be replicated for an isql sessions. Refer to “Forcing replication of DDL commands to the standby database” on page 493 for more information.
Does not copy DML and DDL operations to replicate databases. If the warm standby application also copies data to a replicate database, you must mark tables to be copied to the replicate database with <code>sp_setreptable</code> .	Copies DML operations to standby and replicate databases.
Copies execution of the truncate table command to the standby database. No subscription is needed. Note You can enable or disable replication of truncate table to standby databases with the <code>alter logical connection</code> command. See “Replicating truncate table to standby databases” on page 521 for more information.	<ul style="list-style-type: none"> • If you use Adaptive Server databases, copies execution of truncate table to standby databases. No subscription is needed. • If you use SQL Server databases, does not copy execution of truncate table to standby or replicate databases.
Replication Server uses table name and table owner information to identify a table at the standby database.	<p>If you include the <code>owner_on</code> keywords when you mark a table for replication to the warm standby, Replication Server uses table name and table owner information to identify a table at the standby database.</p> <p>If you include the <code>owner_off</code> keywords when you mark a table for replication to the warm standby, Replication Server uses the table name and “dbo” to identify a table at the standby database.</p>

sp_reptostandby	sp_setreptable
By default, text, image, and rawobject columns are copied to the standby database only if changed.	By default, text and image columns are always copied to the standby database.
If you mark the database tables with <code>sp_reptostandby</code> and <code>sp_setreptable</code> , text, image, and rawobject data may be treated in a different way. Refer to “Replicating text, image, and rawobject data” on page 492 for more information.	If you set the replication status with <code>sp_setrepcol</code> , text, image, and rawobject columns are treated as marked: <code>always_replicate</code> , <code>replicate_if_changed</code> , or <code>do_not_replicate</code> .
The easiest method to use when the active and standby databases are identical. Replication definitions are not required, but can be used to optimize performance.	Replication definitions are not required, but can be used to optimize performance.

Using `sp_reptostandby` to enable replication

Use `sp_reptostandby` to copy DML and supported DDL commands for all user tables to the standby database.

Restrictions and requirements when using `sp_reptostandby`

Consider the following issues when you set up your warm standby application and enable replication with `sp_reptostandby`.

- Both the active and standby databases must be managed by Adaptive Servers and must support RepAgent. Both databases must have the same disk allocations, segment names, and roles. Refer to the *Adaptive Server System Administration Guide* for details.
- The active database name must exist in the standby server. Otherwise, replication of commands or procedures containing the name of that database will fail.
- Replication Server does not support replication of DDL commands containing local variables. You must explicitly define site-specific information for these commands.
- Login information is not replicated to the standby database. Refer to “Making the server user’s IDs match” on page 503 for information about adding login information to the destination Replication Server.
- Some commands not copied to the standby database include:
 - `select into`
 - `update statistics`

- Database or configuration options such as `sp_dboption` and `sp_configure`

“Supported DDL commands and system procedures” and list the DDL commands—Transact-SQL commands and Adaptive Server system procedures—that Replication Server reproduces at the standby database when you enable replication with `sp_reptostandby`. An asterisk marks those commands and stored procedures whose replication is supported for Adaptive Server 12.5 and later.

Supported DDL commands and system procedures

`alter table`
`create default`
`create index`
`create plan*`
`create procedure`
`create rule`
`create schema*`
`create table`
`create trigger`
`create view`
`drop default`
`drop index`
`drop procedure`
`drop rule`
`drop table`
`drop trigger`
`drop view`
`grant`
`installjava*`
`remove java*`
`revoke`
`sp_addalias`
`sp_addgroup`
`sp_addmessage`
`sp_add_qpgroup*`
`sp_adduser`
`sp_addtype`
`sp_bindex`
`sp_bindmsg`
`sp_bindrule`
`sp_changegroup`

```

sp_chgattribute
sp_commonkey
sp_config_rep_agentsp_dropalias
sp_drop_all_qplans*
sp_dropgroup
sp_dropkey
sp_dropmessage
sp_drop_qpgroup*
sp_droptype
sp_dropuser
sp_export_qpgroup*
sp_foreignkey
sp_import_qpgroup*
sp_primarykeysp_procxmode
sp_recompile
sp_rename
sp_rename_qpgroup*
sp_setrepcol
sp_setreplicate
sp_setrepmode
sp_setrepproc
sp_setreptable
sp_unbinddefault
sp_unbindmsg
sp_unbindrule

```

To enable replication of DML and DDL commands, execute `sp_reptostandby` at the Adaptive Server that manages the active database. The syntax is:

```
sp_reptostandby dbname [, 'L1' | 'all' | 'none' ]
```

where *dbname* is the name of the active database and the keywords L1, all, and none set the level of replication support.

L1 represents the level of replication supported by Adaptive Server version 12.5.

Use the all keyword to make sure that schema replication support is always at the highest level available. For example, to set the schema replication support level to that of the latest Adaptive Server version, log in to Adaptive Server and execute this command at the isql prompt:

```
sp_reptostandby dbname, 'all'
```

Then, if the database is upgraded to a later Adaptive Server version with a higher level of replication support, all new features of that version are enabled automatically. Refer to “sp_reptostandby” in Chapter 5, “Adaptive Server Commands and System Procedures,” in the *Replication Server Reference Manual* for complete syntax and user information.

Disabling replication

To turn off data and schema replication, log in to Adaptive Server and enter this command at the isql prompt:

```
sp_reptostandby dbname, 'none'
```

When replication is turned off, Adaptive Server locks all user tables in exclusive mode and saves information about each of them. This process may take some time if there are a large number of user tables in the database.

Use this procedure *only* if you are disabling the warm standby application.

Note If you wish to turn off replication for the current isql session only, use the set replication command. See “Changing replication for the current isql session” on page 493 for more information.

Using sp_setreptable to enable replication

Use sp_setreptable to mark individual tables for replication to replicate or replicate and standby databases. Replication Server copies DML operations on those tables to the standby and replicate databases.

Use sp_setreptable to mark tables for replication to the standby database if:

- You use SQL Server databases, or
- You choose not to use sp_reptostandby.

Using sp_setreptable maintains data, but not schema, consistency between the active and standby databases. sp_setreptable normally does not copy supported DDL commands and procedures to the standby database. You can, however, use the set replication command to force replication of DDL commands for the current isql session. Refer to “Changing replication for the current isql session” on page 493 for more information about set replication.

Using `sp_setreproc` to copy user stored procedures

To copy the execution of a user stored procedure to the standby database, mark the stored procedure for replication with `sp_setreproc`. Procedures marked with `sp_setreproc` are also reproduced at replicate databases if subscriptions have been created for them.

There are two possible scenarios for stored procedure execution in warm standby applications:

- If you have marked the stored procedure for replication with `sp_setreproc`, Replication Server copies execution of the procedure to the standby database. It does *not* copy the effects of the stored procedure to the standby database.
- If you have not marked the stored procedure for replication, Replication copies DML changes effected by the procedure to the standby database, if the affected tables have been marked for replication.

Refer to Chapter 10, “Managing Replicated Functions” for more information about the `sp_setreproc` system procedure.

Replicating tables with the same name but different owners

Adaptive Server and Replication Server allow you to replicate tables with the same name but different owners.

When you mark a database for replication with `sp_reptostandby`, updates are copied automatically to the table of the same name and owner in the standby database.

When you mark a table for replication using `sp_setreptable`, you can choose whether the table owner name is used to select the correct table in the standby database.

- If you set `owner_on`, Replication Server sends the table name and table owner name to the standby database.
- If you set `owner_off`, Replication Server sends the table name and “dbo” as the owner name to the standby database.

Note If you are copying information to a replicate database and have used `sp_setreptable` to set `owner_off`, Replication Server sends the table name to the replicate database. It does not send owner information.

Refer to “Enabling replication with owner_on status” on page 277 for syntax and other information about using `sp_setreptable` to set owner status.

Note If you mark a table with a non-unique name for replication and then create a replication definition for it, you must include owner information in the replication definition. Otherwise, Replication Server will be unable to find the correct table in the replicate or standby database.

Replicating *text*, *image*, and *rawobject* data

If a database is marked with `sp_reptostandby`, the replication status is automatically `replicate_if_changed`, and Adaptive Server logs only *text*, *image*, and *rawobject* columns that have been changed. This ensures that the standby database stays in sync with the active database. You cannot change the replication status of such a table using `sp_setrepcol`.

If a table is marked for replication with `sp_setreptable`, the default replication status is `always_replicate`, and Adaptive Server logs all *text*, *image*, and *rawobject* column data. You can change the replication status of *text*, *image*, and *rawobject* columns in tables marked with `sp_setreptable`. Use `sp_setrepcol` to change the replication status to `replicate_if_changed` or `do_not_replicate`. A column or combination of columns must uniquely identify each row.

If you use replication definitions, the primary key must be a set of columns that uniquely identify each row in the table. You have to make sure that replication status is the same at the Adaptive Server and the Replication Server. If the replication status differs, you must resolve the inconsistencies. Refer to “Resolving inconsistencies in replication status” on page 289 for more information.

When warm standby involves a replicate database

You can copy information from an active database to a standby database and also copy information from the active database to a replicate database. Replication Server must copy a table’s *text*, *image*, and *rawobject* columns to the standby and replicate databases with the same replication status.

Do not change the replication status for the table if you want to copy all *text*, *image*, and *rawobject* columns to the standby and replicate databases. By default, all *text*, *image*, and *rawobject* columns are copied to standby and replicate databases.

If you want to copy only text, image, and rawobject columns that have changed, use `sp_setrepcol` to set the replication status to `replicate_if_changed`.

Changing replication for the current *isql* session

You can use set replication to control replication of DML and/or DDL commands and procedures for an *isql* session.

Execute set replication at the Adaptive Server that manages the active database. The syntax is:

```
set replication ['on' | 'force_ddl' | 'default' | 'off']
```

The default setting is “on.” Default behavior depends on whether or not the database has been marked for replication with `sp_reptostandby`. Table 15-3 describes the default behavior of set replication.

Table 15-3: Default behavior of set replication

If the database has been marked for replication with <code>sp_reptostandby</code>	If the database has not been marked for replication with <code>sp_reptostandby</code>
Replication Server copies DML and supported DDL commands to the standby database for all user tables.	Replication Server copies DML commands to standby and replicate databases for tables marked with <code>sp_setreptable</code> .

Some examples of set replication follow. Refer to “set replication” in Chapter 5, “Adaptive Server Commands and System Procedures” in the *Replication Server Reference Manual* for complete syntax and usage information.

Forcing replication of DDL commands to the standby database

To force replication of all supported DDL commands and system procedures for an *isql* session, enter:

```
set replication 'force_ddl'
```

This command enables replication of DDL commands and system procedures for tables marked with `sp_setreptable`.

To turn off `force_ddl` and return set replication to default status, enter:

```
set replication 'default'
```

Turning off all replication to the standby database

To turn off all replication to the standby database for an *isql* session, enter:

```
set replication 'off'
```

Setting up warm standby databases

Setting up databases for a warm standby application involves three high-level tasks. Each of these tasks may include additional tasks.

- 1 Create a single logical connection that will be used by both the active and standby databases.
- 2 Use Sybase Central or `rs_init` to add the active database to the replication system. You do not need to add the active database if you have designated as the active database a database that was previously added to the replication system.
- 3 Use `sp_reptostandby` or `sp_setreptable` to enable replication for tables in the active database.
- 4 Use Sybase Central or `rs_init` to add the standby database to the replication system, then initialize the standby database.

Before you begin

Before setting up the databases, note that:

- The Replication Server that manages the active and standby databases must be installed and running. A single Replication Server manages both the active and the standby database.
- The Adaptive Servers that contain the active and standby databases must be installed and running. Ideally, these databases should be managed by data servers running on different machines.
- Before you can add a database to the replication system as an active or standby database, it must already exist in the Adaptive Server.

See “Warm standby requirements and restrictions” on page 483 for additional information.

Client application issues

Depending on your client applications and your method of initializing the standby database, you may be suspending transaction processing in the active database until you have initialized the standby database.

If you do not suspend transaction processing, ensure that Replication Server has sufficient stable queue space to hold the transactions that execute while you are loading data into the standby database.

Before you set up the warm standby databases, you should have decided on and implemented a mechanism for switching client applications to the new active database. See “Setting up clients to work with the active data server” on page 516 for more information.

Task one: Creating the logical connection

This section explains how to create the logical connection for the warm standby application. See “Database connections in a warm standby application” on page 481 for more information about logical connections.

This section also explains how to reconfigure RepAgent for the active database, which you must do if the active database was already part of the replication system.

RSM

For instructions on creating a logical connection in Sybase Central, see “Creating a logical connection” in Replication Server’s plug-in help.

Naming the logical connection

When you create the logical connection, use the combination of logical data server name and logical database name, in the form *data_server.database*.

There are two methods for naming the logical connection:

- *If the active database has not yet been added to the replication system* – use a different name for the logical connection than for the active database. Using unique names for the logical and physical connections makes switching the active database more straightforward.
- *If the active database has previously been added to the replication system* – use the *data_server* and *database* names of the active database for the logical connection name. The logical connection inherits any existing replication definitions and subscriptions that reference this physical database.

When you create a replication definition or subscription for a warm standby application, specify the logical connection instead of a physical connection. Specifying the logical connection allows Replication Server to reference the currently active database.

See “Warm standby applications using replication” on page 524 for more information. Also see “Using replication definitions and subscriptions” on page 531.

Procedure for creating the logical connection

Follow these steps to create the logical connection:

- 1 Using a login name with sa permission, log in to the Replication Server that will manage the warm standby databases.
- 2 Execute the create logical connection command:

```
create logical connection to data_server.database
```

The data server name can be any valid Adaptive Server name, and the database name can be any valid database name.

Reconfiguring and restarting RepAgent

If you designate as the active database a database that was previously added to the replication system, the RepAgent thread for the active database shuts down when you create the logical connection.

- 1 Reconfigure RepAgent with `sp_config_rep_agent`, setting the `send_warm_standby_xacts` configuration parameter.
- 2 Restart RepAgent.

For information about how to configure and start RepAgent, refer to “Setting up RepAgent” on page 120. Refer to “`sp_config_rep_agent`” on page 5-10 in the *Replication Server Reference Manual* for more information about the `sp_config_rep_agent` system procedure.

If you are using an LTM as the replication agent, start the LTM using the `-W` command line flag. Refer to “lrm” in Chapter 7, “Programs,” in the *Replication Server Reference Manual* for information about the lrm program.

Task two: Adding the active database

To add a database to the replication system as the active database for a warm standby application, `rs_init`, as described in the Replication Server installation and configuration guides for your platform. Perform the steps described for adding a database to the replication system.

RSM

For instructions on adding an active database in Sybase Central, see “Adding the active database” in Replication Server’s plug-in help.

Note Remember, you do not need to add the active database if you have designated a database that is already part of the replication system as the active database.

Task three: Enabling replication for objects in the active database

You can enable replication for tables in the active database in either of two ways:

- Use `sp_reptostandby` to mark the database for replication, enabling replication of data and supported schema changes.
- Use `sp_setreptable` to mark individual tables for replication of data changes.

Refer to “What information is replicated?” on page 485 for more information about these commands.

- 1 Log in to the Adaptive Server as the System Administrator or as the Database Owner, and:

```
use active_database
```

- 2 Mark database tables for replication, using one of these methods:

- Mark all user tables by executing the `sp_reptostandby` system procedure:

```
sp_reptostandby dbname, [ 'L1' | 'all' ]
```

where *dbname* is the name of the active database, L1 sets the replication level to that of Adaptive Server version 11.5, and all sets the replication level to the current version of Adaptive Server. This method replicates both DML and DDL commands and procedures.

- Mark individual user tables by executing the `sp_setreptable` system procedure for each table that you want to replicate into the standby database:

```
sp_setreptable table_name, 'true'
```

where *table_name* is the name of the table. This method replicates DML commands.

- 3 If you are using the replicated functions feature described in Chapter 10, “Managing Replicated Functions” execute the following system procedure for every stored procedure whose executions you want to replicate into the standby database:

```
sp_setrepproc proc_name, 'function'
```

- 4 If you are using replicated stored procedures associated with table replication definitions, as described in Appendix A, “Asynchronous Procedures,” execute the following system procedure for every such stored procedure whose executions you want to replicate into the standby database:

```
sp_setrepproc proc_name, 'table'
```

Enabling replication for objects added later

Later on, you may add new tables and user stored procedures that you want to replicate to the standby database.

- If you marked the database for replication with `sp_reptostandby`, new tables are automatically marked for replication.
- If you marked database tables for replication to the standby database with `sp_setreplicate`, you must mark each new table that you want to replicate with `sp_setreplicate`.
- You must mark each new user stored procedure that you want to replicate with `sp_setrepproc`.

Task four: Adding the standby database

Use `rs_init` to add the standby database and its RepAgent to the replication system, then you initialize it with data from the active database.

RSM

For instructions on adding the standby database in Sybase Central, see “Adding the standby database” in Replication Server’s plug-in help.

This section explains how to add the standby database to the replication system and prepare it for operation.

This section also describes enabling replication for objects in the standby database and granting permissions to the maintenance user in the standby database. Whether or not you need to perform these steps depends on your method for initializing the standby database.

Before you add the standby database:

- 1 Create the standby database, if it does not already exist.
- 2 Determine how to initialize the standby database.
- 3 Add the standby database maintenance user—if you are using dump and load to initialize the standby database.
- 4 Online the new database using the online database clause before replicating.

Creating the standby database

If it does not already exist, you must create the standby database in the appropriate Adaptive Server, according to your needs.

Refer to the *Adaptive Server System Administration Guide* for details on creating databases.

Determining how to initialize the standby database

You initialize the standby database with data from the active database. To do this, use these Adaptive Server commands and utilities:

- dump and load, or
- bcp, or
- quiesce database ... to *manifest_file* and mount.

Replication Server writes an “enable replication” marker into the active database transaction log when you add the standby database using Sybase Central or `rs_init`. Adaptive Server writes a dump marker into the active database transaction log when you perform a dump operation.

If you do not suspend transaction processing during initialization:

- Choose the “dump marker” option in Sybase Central or `rs_init`, and use the dump and load commands.

If you suspend transaction processing during initialization:

- Do not choose the “dump marker” option in Sybase Central or `rs_init`, and use the dump and load commands, or
- Use `bcp`, or
- Use quiesce database ... to *manifest_file* and mount.

Table 15-4 summarizes each of the initialization methods and the role of these markers.

Table 15-4: Issues in initializing the standby database

Issue	Use dump and load with “dump marker”	Use dump and load without “dump marker”	Use bcp	Use mount
Working with client applications.	Use if you can not suspend transaction processing for client applications.	Use if you can suspend transaction processing for client applications.		Use if you can suspend transaction processing for client applications.
When does Replication Server begin replicating into the standby database?	Replication Server starts replicating into the standby database from the first dump marker after the enable replication marker.	Replication Server starts replicating into the standby database from the enable replication marker.		Replication Server starts replicating into the standby database from the enable replication marker.
Creating maintenance user login names and making sure all user IDs match.	Add the login name for the standby database maintenance user in both the active Adaptive Server and the standby Adaptive Server, and ensure that the server user’s IDs match. (You create login names in the active Adaptive Server because using dump and load to initialize the standby database with data from the active database overrides any previous contents of the standby database with the contents of the active database.)		When you add the standby database, Sybase Central or rs_init adds the maintenance user login name and user in the standby Adaptive Server and the standby database.	Add the login name for the standby database maintenance user in both the active and standby Adaptive Servers. Ensure that the server user’s IDs match. (You create login names in the active Adaptive Server because using mount to initialize the standby database with data from the active database overrides any previous contents of the standby database with the contents of the active database.)
Initializing standby database.	Use dump and load to transfer data from the active database to the standby database. You can use database dumps and/or transaction dumps.		Use bcp to copy each replicated table from the active database to the standby database.	Use quiesce database ... to <i>manifest_file</i> and mount database to transfer data from the active database to the standby database.
Active database connection state.	The connection to the active database does not change.	Replication Server suspends the connection to the active database.		Replication Server suspends the connection to the active database.

Issue	Use dump and load with “dump marker”	Use dump and load without “dump marker”	Use bcp	Use mount
Resuming connections.	Resume connection to the standby database.	Resume connections to the active and standby databases; resume transaction processing in the active database.		Resume connections to the active and standby database; resume transaction processing in the active database.

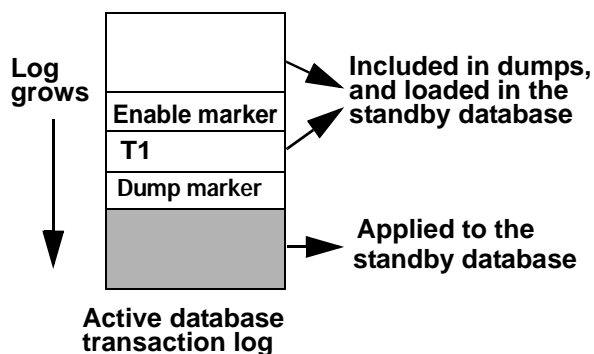
If you do not suspend transaction processing

If you do not suspend transaction processing for the active database while initializing the standby database, choose the “dump marker” option when you add the standby database. Then initialize the standby database by using the dump and load commands.

Replication Server starts replicating into the standby database from the first dump marker after the enable replication marker in the transaction log of the active database.

In Figure 15-2, transaction T1, executed after you added the standby database, appears after the enable replication marker in the log. T1 is included in dumps, so it is present in the standby database after you have loaded the dumps. Replication Server does not need to replicate it into the standby database.

Figure 15-2: Using dump and load with dump marker



Transactions can be executed in the active database between the time the enable replication marker is written and the time the data in the active database is dumped.

You can load the last full database dump and any subsequent transaction dumps into the standby database until both markers have been received and the standby database is ready for operation. Then, optionally, you can use a final transaction dump of the active database to bring the standby database up to date. Any transactions not included in dumps will be replicated.

If you suspend transaction processing

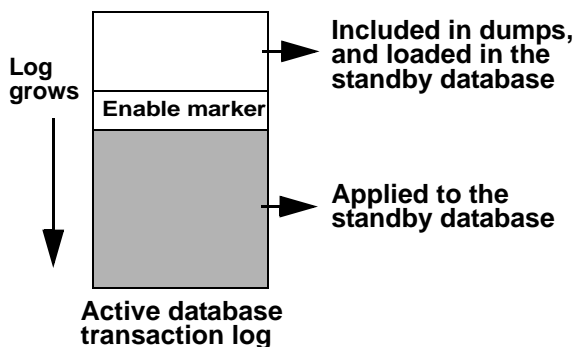
Replication Server does not replicate transactions from the active to the standby database until it has received both the enable replication marker and the first subsequent dump marker. After receiving both markers, Replication Server starts executing transactions in the standby database.

See Table 15-4 for more information about this method.

If you suspend transaction processing for the active database while initializing the standby database, do not choose the “dump marker” option when you add the standby database. You can initialize the standby database by using the dump and load commands, by using bcp, or by using mount.

Replication Server starts replicating into the standby database from the enable replication marker in the transaction log of the active database. No transactions occur after the enable replication marker, because client applications are suspended.

Figure 15-3: Using dump and load without dump marker, or using bcp



As shown in Figure 15-3, no transactions are executed in the active database between the time the enable replication marker is written and the time the data in the active database is dumped using the dump command, or copied using bcp or mount.

You can load the last full database dump or the last set of replicated tables copied with bcp into the standby database until the standby database receives the enable replication marker.

After receiving this marker, Replication Server starts executing transactions in the standby database.

Adding the standby database maintenance user

If you plan to initialize the standby database using the dump and load commands, with or without the “dump marker” option, you must create the maintenance user login name for the standby database in both the standby and the active data servers. Do this before you add the standby database.

Both Sybase Central and `rs_init` automatically add the active database maintenance user in the active data server when you add the active database.

Making the server user's IDs match

Within each data server, the server user's ID (suid) for each login name must be the same in the `syslogins` table in the master database and the `sysusers` table in each user database. This must be true for the active and standby databases in a warm standby application. The server user's ID and role settings must also be the same in the `syslogins` and `sysloginroles` tables in the master database.

Use one of these three methods to make the server user's IDs match:

- Add all login names, including maintenance user names, to both Adaptive Servers in the same order. Adaptive Server assigns server user's IDs sequentially, so the server user's IDs for all login names will match.
- After loading the dump into the standby, reconcile the `sysusers` table in the standby database with the `syslogins` table in the master database of the standby Adaptive Server.
- Maintain a master Adaptive Server with all of your login names and copy the `syslogins` table from the master database for the master Adaptive Server to all newly created Adaptive Servers.

Adding the maintenance user

To add the maintenance user login name for the standby database to both the standby and the active data servers:

- 1 In the standby data server, execute the `sp_addlogin` system procedure to create the maintenance user login name.

Refer to the *Adaptive Server System Administration Guide* for more information about using `sp_addlogin`.

- 2 In the active data server, execute `sp_addlogin` to create the same maintenance user login name that you created in the standby data server.

When you set up the standby database using the dump and load commands, the `sysusers` table is loaded into the standby database along with the other data from the active database.

Adding the standby database to the replication system

To add the standby database to the replication system:

- 1 Suspend transaction processing in the active database, if appropriate for your client applications and your method of initializing the standby database.

You must use dump and load with the “dump marker” method if you do not suspend transaction processing.

- 2 Use Sybase Central or `rs_init` to add the standby database to the replication system. Perform the steps described for adding a database to the replication system.

- 3 To monitor the status of the logical connection at any time, enter:

```
admin logical_status, logical_ds, logical_db
```

The Operation in Progress and State of Operation in Progress output columns indicate the standby creation status.

- 4 If you are initializing the standby database using dump and load, use the dump command to dump the contents of the active database, and load the standby database. For example:

```
dump database active_database to dump_device
load database standby_database from dump_device
```

- 5 If you have already loaded a previous database dump and subsequent transaction dumps, you can just dump the transaction log and load it into the standby database. For example:

```
dump transaction active_database to dump_device
load transaction standby_database from dump_device
```

- 6 After completing load operations, bring the standby database online:

```
online database standby_database
```

Refer to the *Adaptive Server Reference Manual* for help with using the dump and load commands and the online database command.

- 7 Initialize the standby database. Use `bcp` or `quiesce ... to manifest_file` and `mount`.

- To initialize the standby database using `bcp`, copy each of the replicated tables in the active database to the standby database.

You must copy the `rs_lastcommit` table, which was created when you added the active database to the replication system.

Refer to the Adaptive Server utility programs manual for help with using the bcp program.

- To initialize the standby database using `quiesce ... to manifest_file` and `mount`, quiesce the database and create the manifest file. Make a copy of both the database and log devices. Mount the devices on the standby database.
- 8 If you initialized the standby database by using `dump and load` without the “dump marker” method, or by using `bcp`, or by using `quiesce database ... to manifest_file` and `mount`, Replication Server suspended the connection to the active database. Resume the connection by executing the following command in the Replication Server:

```
resume connection to active_ds.active_db
```

- 9 Regardless of your method for initializing the standby database, you must resume the connection to the standby database by executing the following command:

```
resume connection to standby_ds.standby_db
```

- 10 Resume transaction processing in the active database, if it was suspended.

Using a blocking command for standby creation

In Replication Server, the `wait for create standby` command is a blocking command. It tells Replication Server not to accept commands until the standby database is ready for operation. You can use this command in a script that creates a standby database. The syntax is:

```
wait for create standby for logical_ds.logical_db
```

Enabling replication for objects in the standby database

To be ready to switch to the standby database, replication must be enabled for the tables and stored procedures in the standby database that you want to replicate into the new standby database after the switch.

- If you initialized the standby database using the `dump and load` or `mount` commands, the tables and stored procedures in the standby database will have the same replication settings as the active database.
- If you initialized the standby database using `bcp`, enable replication for these objects by using `sp_setreptable` or `sp_reptostandby`, and `sp_setrepproc`. To do this, adapt the procedure under “Task three: Enabling replication for objects in the active database” on page 497 to enable replication for objects in the standby database.

Enabling replication for objects added later

Later on, you may add new tables and user stored procedures that you want to replicate to the new standby database.

- If you marked the standby database for replication with `sp_reptostandby`, any new tables are automatically marked for replication.
- If you marked individual database tables for replication to the new standby database with `sp_setreplicate`, you must mark each new table that you want to replicate with `sp_setreplicate`.
- You must mark each new user stored procedure that you want to replicate with `sp_setrepproc`.

Granting permissions to the maintenance user

After adding the standby database, you must grant the necessary permissions to the maintenance user.

To grant permissions:

- 1 Log in to the Adaptive Server as the System Administrator or as the Database Owner, and enter:

```
use standby_database
```

- 2 Grant `replication_role` to the maintenance user. `replication_role` ensures that the maintenance user can execute `truncate table` at the standby database.

```
sp_role "grant", replication_role, maintenance_user
```

- 3 Execute this command for each table:

```
grant all on table_name to maintenance_user
```

Switching the active and standby databases

This section contains information about switching to the standby database when the active database fails or when you want to perform maintenance on the active database.

Determining if a switch is necessary

Determining when it is necessary to switch from the active to the standby database depends on the requirements of your applications.

In general, you should not switch when the active data server experiences a transient failure. A transient failure is a failure from which the Adaptive Server recovers upon restarting with no need for additional recovery steps. You probably should switch if the active database will be unavailable for a long period of time.

Determining when to switch depends on issues such as how much recovery the active database requires, to what degree the active and standby databases are in sync, and how much downtime your users or applications can tolerate.

You may also want to switch the roles of the active and standby databases to perform planned maintenance on the active database or its data server.

Before switching active and standby databases

Figure 15-4 illustrates a warm standby application for a database that does not participate in the replication system other than through the activities of the warm standby application itself. Figure 15-4 represents the warm standby application in normal operation, before you switch the active and standby databases.

Figure 15-4: Warm standby application example—before switching

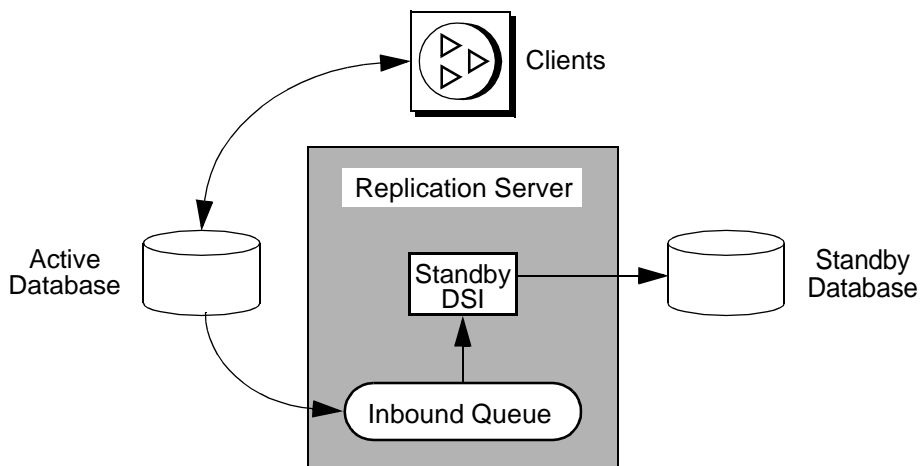


Figure 15-4 adds internal detail to Figure 15-1, to show that:

- Replication Server writes transactions received from the active database into an inbound message queue.

See “Distributed concurrency control” on page 48 for more information about inbound and outbound queues.

- This inbound queue is read by the DSI thread for the standby database, which executes the transactions in the standby database.

Messages received from the active database cannot be truncated from the inbound queue until the standby DSI thread has read them and applied them to the standby database.

In this example, transactions are simply replicated from the active database into the standby database. The logical database itself does not:

- Contain primary data that is replicated to replicate databases or remote Replication Servers, or
- Receive replicated transactions from another Replication Server

See “Warm standby applications using replication” on page 524 for information about warm standby applications for a primary or replicate database.

Internal switching steps

When you switch active and standby databases, here is what Replication Server does:

- 1 Issues log suspend against the active and standby RepAgent connections.
- 2 Reads all messages left in the inbound queue and applies them to the standby database and, for subscription data or replicated stored procedures, to outbound queues.

All committed transactions in the inbound queue must be processed before the switch can complete.

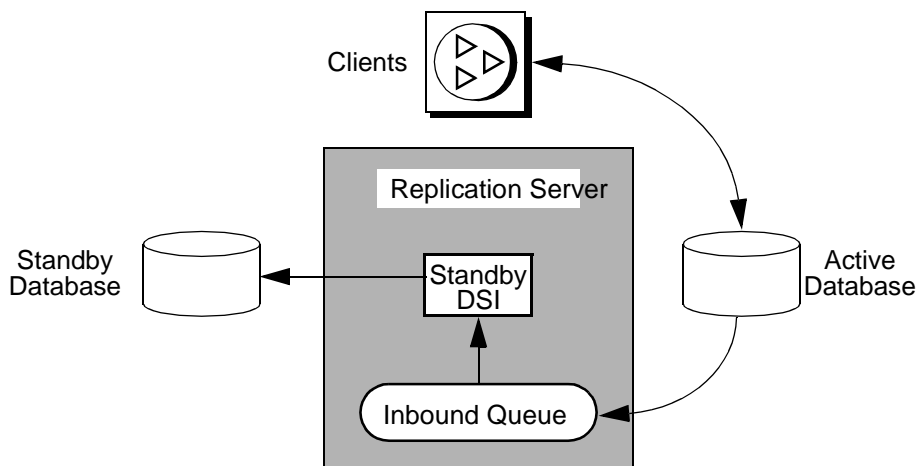
- 3 Suspends the standby DSI.
- 4 Enables the secondary truncation point in the new active database.
- 5 Places a marker in the transaction log of the new active database. Replication Server uses this marker to determine which transactions to apply to the new standby database and to any replicate databases.

- 6 Updates data in the RSSD pertaining to the warm standby databases.
- 7 Resumes the connection for the new active database, and resumes log transfer for the new active database so that new messages can be received.

After switching active and standby databases

After you have switched the roles of the active and standby databases, the replication system will have changed, as shown in Figure 15-5:

Figure 15-5: Warm standby application example—after switching



- The previous standby database is the new active database. Client applications will have switched to the new active database.
- The previous active database, in this example, becomes the new standby database. Messages for the previous active database are queued for application to the new active database.

Note RepAgent for the previous active database has shut down. RepAgent for the new active database has started.

Making the switch

To switch from the active to the standby database:

- Disconnect client applications from the active database if they are still using it
- In Replication Server, switch the active and standby databases
- Restart client applications with the new active database
- Start RepAgent for the new active database
- Determine whether to drop the old active database or use it as the new standby database

The following sections contain the procedures for these tasks.

RSM

For instructions about switching to the active database in Sybase Central, see “Switching the active and standby databases” in Replication Server’s plug-in help.

Disconnect client applications from the active database

Before you switch to the standby database, you must stop clients from executing transactions in the active database. If the database failed, of course, clients cannot execute transactions. However, you may need to take steps to prevent them from updating that database after it is back online.

See “Setting up clients to work with the active data server” on page 516 for more information about client application issues.

Procedure for switching the active and standby databases

Before switching, you must implement a method for switching clients, as described in “Setting up clients to work with the active data server” on page 516.

Follow these steps to switch the active and standby databases for a logical connection:

- 1 At the Adaptive Server of the active database, ensure that the RepAgent is shut down. Otherwise, use `sp_stop_rep_agent` to shut down the RepAgent.
- 2 At the Replication Server, enter:

```
switch active for logical_ds.logical_db
to data_server.database
```

data_server.database is the new active database.

See “Internal switching steps” on page 508 for information on what Replication Server does when you switch.

- 3 To monitor the progress of a switch, you can use the `admin logical_status` command:

```
admin logical_status, logical_ds, logical_db
```

The Operation in Progress and State of Operation in Progress output columns indicate the switch status.

- 4 When the active database switch is complete, you must restart RepAgent for the new active database:

```
sp_start_rep_agent dbname
```

Note If Replication Server stops in the middle of switching, the switch resumes after you restart Replication Server.

Using a blocking command for switch active

In Replication Server, the `wait for switch` command is a blocking command. It tells Replication Server to wait until the standby database is ready for operation. You can use this command in a script that switches the active database. The syntax is:

```
wait for switch for logical_ds.logical_db
```

Monitoring the switch

You can use `admin logical_status` to check for replication system problems that prevent the switch from proceeding. Such problems may include a full transaction log for the standby database or a suspended standby DSI. If you cannot resolve the problems, you can abort the switch using the `abort switch` command.

The Operation in Progress and State of Operation in Progress output columns indicate the switch status.

For example, suppose the `admin logical_status` command persistently returns one of the following messages in its State of Operation in Progress output column:

```
Standby has some transactions that have not been applied  
or  
Inbound Queue has not been completely read by  
Distributor
```

These messages may indicate a problem that you cannot resolve, in which case you may choose to abort the switch. You can use `admin who` commands to obtain more information about the state of the switching operation.

See “Commands for monitoring warm standby applications” on page 515 for more information.

Aborting a switch

Unless Replication Server has proceeded too far in switching the active and standby databases, you can abort the process by using the `abort switch` command:

```
abort switch for logical_ds.logical_db
```

If the `abort switch` command cancels the `switch active` command successfully, you may have to restart the RepAgent for the active database.

You cannot cancel the `switch active` command after it reaches a certain point. If this is the case, you must wait for the `switch active` command to complete, then use it again to return to the original active database.

Restart client applications

When the `admin logical_status` command indicates that there is no operation in progress, or when the `wait for switch` command returns an `isql` prompt, you can restart client applications in the new active database.

Client applications must wait until Replication Server’s switch to the new active database is complete before they begin executing transactions in the new active database. You should provide an orderly method for moving clients from the old active database to the new active database. See “Setting up clients to work with the active data server” on page 516 for more information.

Resolving paper-trail transactions

If the old active database failed, determine if any transactions were not transmitted to the new active database. Such transactions are called **paper-trail transactions** if there is an external record of their execution.

When you switch from an active to a standby database, all committed transactions in the inbound queue are applied to the new active database before the switch is complete. However, it is possible that some transactions that committed at the active database before the failure were not received by Replication Server and, therefore, were not applied to the standby database.

When you switch the active and standby databases, you can re-execute the paper-trail transactions in the new active database. If there are dependencies, you may need to re-execute the paper-trail transactions before you allow new transactions to execute. Be sure to execute the paper-trail transactions using the original client's login name, not the maintenance user login name.

If you bring the old active database online as the new standby database, you must first reverse the paper-trail transactions so they will not be duplicated in the standby database.

Manage the old active database

After you have switched to the new active database, you must decide what to do with the old active database. You can:

- Bring the database online as the new standby database and resume the connection so that Replication Server can apply new transactions, or
- Drop the database connection using the drop connection command, and add it again later as the new standby database. If you drop the database, any queued messages for the database are deleted. Refer to “drop connection” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual*.

Bringing the old active database online as the new standby

If the old active database is undamaged, you can bring it back online as the new standby database by entering:

```
resume connection to data_server.database
```

where *data_server.database* is the physical database name of the old active database.

You may need to resolve paper-trail transactions in the database in order to avoid duplicate transactions. Depending on your applications, you may need to do this before you bring the old active database back online as the new standby database.

Because paper-trail transactions must be re-executed in the new active database, you must prepare the new standby database so that it can receive the transactions again when they are delivered through the replication system.

To resolve the conflicts, you can:

- Undo or reverse the duplicate transactions in the new standby database, or
- Ignore the duplicate transactions and deal with them later.

Monitoring a warm standby application

This section describes methods you can use to monitor a warm standby application.

Replication Server log file

You can read the Replication Server log file for messages pertaining to warm standby operations. This section discusses log messages you will see when you add the standby database.

Standby connection created

These are examples of the messages that Replication Server writes while creating the physical connection for a standby database:

```
I. 95/11/01 17:47:50. Create starting : SYDNEY_DS.pubs2
I. 95/11/01 17:47:58. Placing marker in TOKYO_DS.pubs2 log
I. 95/11/01 17:47:59. Create completed : SYDNEY_DS.pubs2
```

In these examples, SYDNEY_DS is the standby data server and TOKYO_DS is the active data server.

When you create the physical connection for the standby database, Replication Server writes an “enable replication” marker in the active database transaction log. The standby DSI ignores all transactions until it has received this marker. If, however, you chose the “dump marker” option, the standby DSI continues to ignore messages until it encounters the next dump marker in the log.

When the appropriate marker arrives at the standby database from the active database RepAgent, the standby DSI writes a message in the Replication Server log file and then begins executing subsequent transactions in the standby database.

In the example messages above, Replication Server has created the connection for the standby database, SYDNEY_DS.pubs2, and suspended its DSI thread. At this point, the Database Administrator dumps the contents of the active database, TOKYO_DS.pubs2, and loads it into the standby database.

Standby connection resumed after initialization

After the Database Administrator has loaded the dump into the standby database and resumed the connection to the standby database, the standby DSI begins processing messages from the active database. Replication Server writes in its log messages similar to this:

```
I. 95/11/01 18:50:34. The DSI thread for database 'SYDNEY_DS.pubs2' is started.
I. 95/11/01 18:50:41. Setting LTM truncation to 'ignore' for SYDNEY_DS.pubs2 log
I. 95/11/01 18:50:43. DSI for SYDNEY_DS.pubs2 received and processed Enable
  Replication Marker. Waiting for Dump Marker
I. 95/11/01 18:50:43. DSI for SYDNEY_DS.pubs2 received and processed Dump
  Marker. DSI is now applying commands to the Standby
```

When you see the final message in the log file, the warm standby database creation process has completed.

Commands for monitoring warm standby applications

Use the admin commands to monitor the status of a warm standby application. Refer to Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information about these commands.

admin logical_status

The admin logical_status command tells you:

- How the addition of a standby database or the switching between active and standby databases is progressing.
- Whether the active or standby database connection is suspended.
- Whether the standby DSI is ignoring messages. The standby DSI ignores messages while it waits for a marker to arrive in the transaction stream from the active database.

admin who, dsi

The admin who, dsi command provides another method to check the status of the standby DSI. The IgnoringStatus output column contains either:

- “Applying” – if the DSI is applying messages to the standby database, or
- “Ignoring” – if the DSI is waiting for a marker.

admin who, sqm

The `admin who, sqm` command provides information about the state of stable queues. In a warm standby application, the inbound queue is read by the Distributor thread, if you have not disabled it, and by the standby DSI thread. Replication Server cannot delete messages from the inbound queue until both threads have read and delivered them.

If Replication Server is not deleting messages from the inbound queue, you can use the `admin who, sqm` command to investigate the problem. The command tells you how many threads are reading the queue and the minimum deletion point in the queue.

admin sqm_readers

The `admin sqm_readers` command monitors the read and delete points of the individual threads that are reading the inbound queue. If the inbound queue is not being deleted, `admin sqm_readers` will help you find the thread that is not reading the queue.

The `admin sqm_readers` command takes two parameters: the queue number and the queue type for the logical connection.

You can find the queue number and queue type in the Info column of the `admin who, sqm` command output: the queue number is the 3-digit number to the left of the colon, while the queue type is the digit to the right of the colon.

Queue type 1 is an inbound queue. Queue type 0 is an outbound queue. The inbound queue for a logical connection can be read by more than one thread. For example, to find out about the threads reading inbound queue number 102, execute `admin sqm_readers` as follows:

```
admin sqm_readers, 102, 1
```

Setting up clients to work with the active data server

When you switch the active and standby databases in Replication Server using the `switch active` command, Replication Server does not switch client applications to the new active data server and database automatically. You must devise a method to switch client applications. This section describes three sample methods for setting up client applications to connect to the currently active data server. You can create:

- Two interfaces files
- An interfaces file entry with a symbolic data server name for client applications
- A mechanism that automatically maps the client application data server connections to the currently active data server

You must implement your method before you set up the warm standby databases.

Regardless of your method for switching applications, do not modify the interfaces file entries used by Replication Server.

Two interfaces files

With this method, you set up two interfaces files, one for the client applications and one for Replication Server. When you switch the clients, you can modify their interfaces file entry to use the host name and port number of the data server with the new active database.

Symbolic data server name for client applications

With this method, you create an interfaces file entry with a symbolic data server name for client applications.

The interfaces file might contain entries like these:

Table 15-5: Symbolic data server name in interfaces file

	Data server name	Host name	Port number
Client applications	CLIENT_DS	<i>machine_1</i>	2800
Active database	TOKYO_DS_X	<i>machine_1</i>	2800
Standby database	TOKYO_DS_Y	<i>machine_2</i>	2802

You could create an interfaces entry for a data server named CLIENT_DS. Client applications would always connect to CLIENT_DS. The CLIENT_DS entry would use the same host name and port number as the data server with the active database.

Replication Server connects to the same host name and port number as the client applications but uses a different data server name. In this example, Replication Server would switch between the TOKYO_DS_X and TOKYO_DS_Y data servers.

After switching the active database, you would change the CLIENT_DS interfaces entry to the host name and port number of the data server with the new active database—in this example, *machine_2* and port number 2802.

Map client data server to currently active data server

With this method, you create a mechanism, such as an intermediate Open Server application, that automatically maps the client application data server connections to the currently active data server.

Refer to Open Server documentation, such as the *Open Server Server-Library/C Reference Manual*, for more information about how to create such an Open Server application.

Altering warm standby database connections

This section describes options for reconfiguring or modifying the logical database connection and the physical database connections. Under ordinary circumstances, if you set up a warm standby application through the usual procedure, the default settings will work correctly.

Altering logical connections

Use the alter logical connection command to modify parameters that:

- Affect logical connections
- Enable or disable the Distributor thread
- Enable or disable the replication of truncate table to the standby database

Changing parameters affecting logical connections

To update parameters that affect logical connections, log in to the source Replication Server and, at the isql prompt, enter:

```
alter logical connection
to logical_ds.logical_db
set logical_database_param to 'value'
```


where *logical_ds* is the data server name for the logical connection, *logical_db* is the database name for the logical connection, *logical_database_param* is a logical database parameter, and *value* is a character string setting for the parameter.

New settings take effect immediately.

Warning! You should reset the logical connection parameters *materialization_save_interval* and *save_interval* *only* when there is a serious lack of stable queue space. Resetting them (from strict to a given number of minutes) may lead to message loss at the standby database.

Table 15-6 displays the configuration parameters that affect logical database connections.

Table 15-6: Configuration parameters affecting logical connections

logical_database_param	value
materialization_save_interval	Materialization queue save interval. This parameter is only used for standby databases in a warm standby application. Default: "strict" for standby databases
replicate_minimal_columns	Specifies whether Replication Server should send all replication definition columns for all transactions or only those needed to perform update or delete operations at the standby database. Values are "on" and "off." Replication Server uses this value in standby situations only when a replication definition does not contain a "send standby" parameter. Otherwise, Replication Server uses the value of the "replicate minimal columns" or "replicate all columns" parameter in the replication definition. Default: on
save_interval	The number of minutes that the Replication Server saves messages after they have been successfully passed to the destination data server. See "Save interval for recovery" on page 633 for details. Default: 0 minutes

logical_database_param	value
send_standby_repdef_cols	<p>Specifies which columns Replication Server should send to the standby database for a logical connection. Overrides “send standby” options in the replication definition that tell Replication Server which table columns to send to the standby database. Values are:</p> <ul style="list-style-type: none">• on – send only the table columns that appear in the matching replication definition. Ignore the “send standby” option in the replication definition.• off – send all table columns to the standby. Ignore the “send standby” option in the replication definition.• check_repdef – send all table columns to the standby based on “send standby” option. <p>Default: check_repdef</p>

Disabling the Distributor thread

If you do not replicate data from the active database into databases other than the standby database, Replication Server does not need a Distributor thread for the logical connection. You can disable the Distributor thread to save Replication Server resources.

To disable the Distributor thread, you must first drop any subscriptions for the data in the logical database. Then execute alter logical connection at the Replication Server:

```
alter logical connection
  to logical_ds.logical_db
  set distribution off
```

If you decide later to replicate data out of the active database, you can use this command to reenable the Distributor thread.

Warning! If you disable the Distributor thread and then drop the standby database from the replication system, no Replication Server threads will be left to read the inbound queue from the active database. The inbound queue will continue to fill until you either add another standby database, set distribution to “on” for the logical connection, or drop the active database from the replication system.

Replicating *truncate table* to standby databases

Replication Server copies execution of *truncate table* to warm standby databases. The active and standby databases must be Adaptive Server version 11.5 or later to support this feature.

To enable or disable replication of *truncate table*, log in to the source Replication Server and enter:

```
alter logical connection
  to logical_ds.logical_db
  set send_truncate_table to {on | off}
```

If your warm standby application was created *before* you upgraded or installed Replication Server version 11.5 or later, Replication Server does not copy *truncate table* to the standby database unless you enable this feature with *alter logical connection*. To preserve compatibility with existing warm standby applications, the default setting is “off.”

If your warm standby application was created *after* you upgraded or installed Replication Server version 11.5 or later, Replication Server automatically copies *truncate table* to the standby database unless you disable this feature with *alter logical connection*. The default setting is “on.”

Altering physical connections

Use the *alter connection* command at the source Replication Server to modify parameters that affect physical connections for warm standby applications:

```
alter connection to data_server.database
  set database_param to 'value'
```

where *data_server* is the destination data server, *database* is the database the data server manages, *database_param* is a parameter that affects the connection and *value* is a setting for *database_param*.

You must suspend the connection before altering it; then, after executing *alter connection*, you resume the connection for new parameter settings to take effect. Refer to “Altering database connections” on page 177 for more information.

Configuring triggers in the standby database

By default, the standby DSI thread executes a set triggers off Adaptive Server command when it logs in to a standby database. This prevents Adaptive Server from firing triggers for the replicated transactions, thereby preventing duplicate updates in the standby database.

You can alter the default behavior by using the alter connection command to configure a connection to fire or not fire triggers. To do this, set the `dsi_keep_triggers` configuration parameter to “on” or “off.” The default `dsi_keep_triggers` setting for all connections except standby databases is “on.”

Configuring replication in the standby database

The `dsi_replication` configuration parameter specifies whether or not transactions applied by the DSI are marked in the transaction log as being replicated. It must be set to “on” for the active replicate database. By default, it is set to “off” for the standby database and set to “on” for all other databases.

When `dsi_replication` is set to “off,” the DSI executes set replication off in the 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, are not replicated further (except if there is a standby database), setting this parameter to “off” where appropriate writes less information into the transaction log.

Use `admin who, dsi` to see how this parameter is set for a connection.

Changing configuration parameters in the standby database

When you create the standby database, the following configuration parameters, if they are set for the active database, are copied from the active database to the standby database:

Table 15-7: Configuration parameters copied to standby database

batch	batch_begin	command_retry
db_packet_size	dsi_charset_convert	dsi_cmd_batch_size
dsi_cmd_separator	dsi_fadeout_time	dsi_keep_triggers
dsi_large_xact_size	dsi_max_cmds_to_log	dsi_max_text_to_log
dsi_num_large_xact_threads	dsi_num_threads	dsi_replication
dsi_serialization_method	dsi_sql_data_style	dsi_sqt_max_cache_size
dsi_xact_group_size	dsi_xact_in_group	dump_load
parallel_dsi		

You can change the setting of any of these configuration parameters. See Chapter 7, “Managing Database Connections” for more information.

Dropping logical database connections

If you are dismantling a warm standby application, you may need to remove a logical database from the replication system. To do this, drop the standby database, then execute the drop logical connection command. Before you execute the command, you must drop the standby database. See “Dropping database connections” on page 192 for information about dropping physical database connections.

The syntax for drop logical connection is:

```
drop logical connection to data_server.database
```

data_server and *database* represent the logical data server and logical database.

For example, to drop the connection to the pubs2 logical database in the LDS logical data server, enter:

```
drop logical connection to LDS.pubs2
```

Dropping a logical database from the ID Server

When a warm standby application exists in the replication system, logical databases, along with physical databases, data servers, and Replication Servers, are listed in the `rs_idnames` system table in the RSSD for the ID Server. Occasionally, it may be necessary to remove the entry for a logical database from this system table.

For example, if a drop logical connection command fails, you may have to force the ID Server to delete from the `rs_idnames` system table the row that corresponds to the logical database. Logical database connections show an “L” in the `ltype` column.

The `sysadmin dropldb` command logs in to the ID Server and deletes the entry for the specified logical database. The syntax is:

```
sysadmin dropldb, data_server, database
```

data_server and *database* refer to the logical data server and the logical database names.

You must have `sa` permission to execute any `sysadmin` command.

Warm standby applications using replication

This section describes warm standby applications that involve replication, where the logical database serves as a primary or replicate database in the replication system.

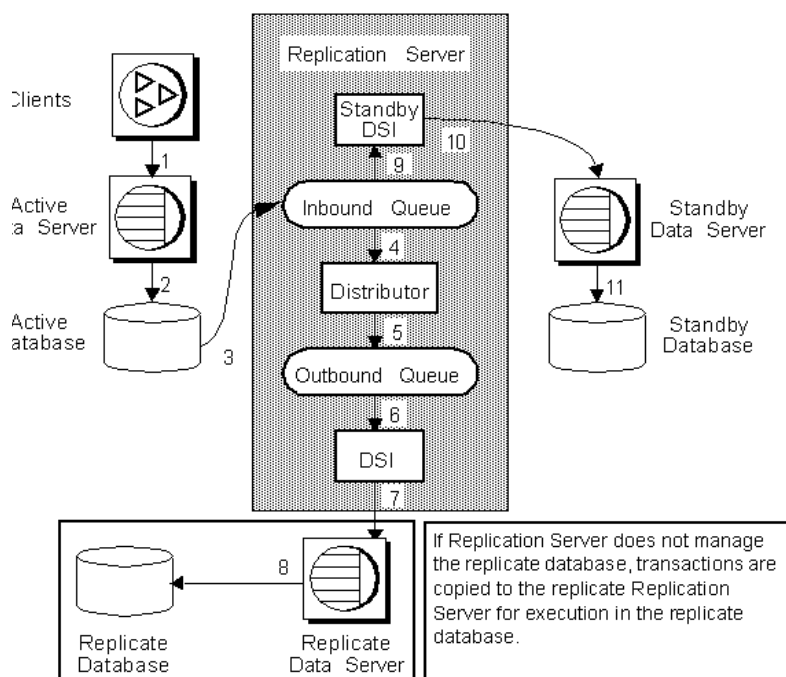
Also see “Using replication definitions and subscriptions” on page 531.

Warm standby application for a primary database

Figure 15-6 illustrates a warm standby application for a primary database. In this example, one Replication Server manages three databases:

- The active database for a logical primary database,
- The standby database for a logical primary database, and
- A replicate database that has subscriptions for the data in the logical primary database.

In this example, a single Replication Server manages both the primary and replicate databases. In other instances, different Replication Servers may manage the primary and replicate databases.

Figure 15-6: Warm standby application for a primary database

The numbers in Figure 15-6 indicate the flow of transactions from client applications through the replication system in a warm standby application for a primary database.

From client applications to inbound queue

In Figure 15-6, numbers 1 through 3 trace transactions from clients to an inbound queue in the Replication Server:

- Clients execute transactions in the active primary data server.
- The active primary data server updates the active primary database.
- The RepAgent thread for the active primary database reads transactions for replicated data in the database log. It forwards the transactions to the Replication Server, which writes them into an inbound queue.

All transactions for replicated data, including those executed by the maintenance user, are sent to the Replication Server for application in the standby database.

From inbound queue to replicate database

In Figure 15-6, numbers 4 through 8 trace transactions from the inbound queue to the replicate database:

- The Distributor thread reads transactions from the inbound queue.
- The Distributor thread processes transactions against subscriptions and writes replicated transactions into an outbound queue.

Transactions executed by the maintenance user, which are always replicated into the standby database (because you set the `send_warm_standby_xacts` parameter when you configure RepAgent with `sp_config_rep_agent`), are not replicated to replicate databases unless you also set the `send_maint_xacts_to_replicate` parameter for RepAgent.

- A DSI thread reads transactions from the outbound queue.
- The DSI thread executes the transactions in the replicate data server.
- The replicate data server updates the replicate database.

If the transactions are to be replicated to a database managed by a different Replication Server, they are written into an RSI outbound queue managed by an RSI thread instead of a DSI thread. The RSI thread delivers the transactions to the other Replication Server.

From inbound queue
to standby database

In Figure 15-6, numbers 9 through 11 trace transactions from the inbound queue to the standby database for the logical primary database:

- The standby DSI thread reads transactions from the inbound queue.
- The standby DSI thread executes transactions in the standby data server.
- The standby data server updates the standby database.

The inbound queue is read by the standby DSI and the Distributor. The two threads do their work concurrently. Messages cannot be truncated from the inbound queue until both threads have read them and delivered them to their destination. The messages remain in the queue until the DSI has applied them to the standby database and, if there are subscriptions or replicated stored procedure executions, the Distributor has written them to the outbound queue.

Depending on your replication system, the transactions may be replicated into the standby database before the replicate database. However, Replication Server guarantees that the standby primary database and replicate databases will be kept in sync with the active primary database.

Warm standby application for a replicate database

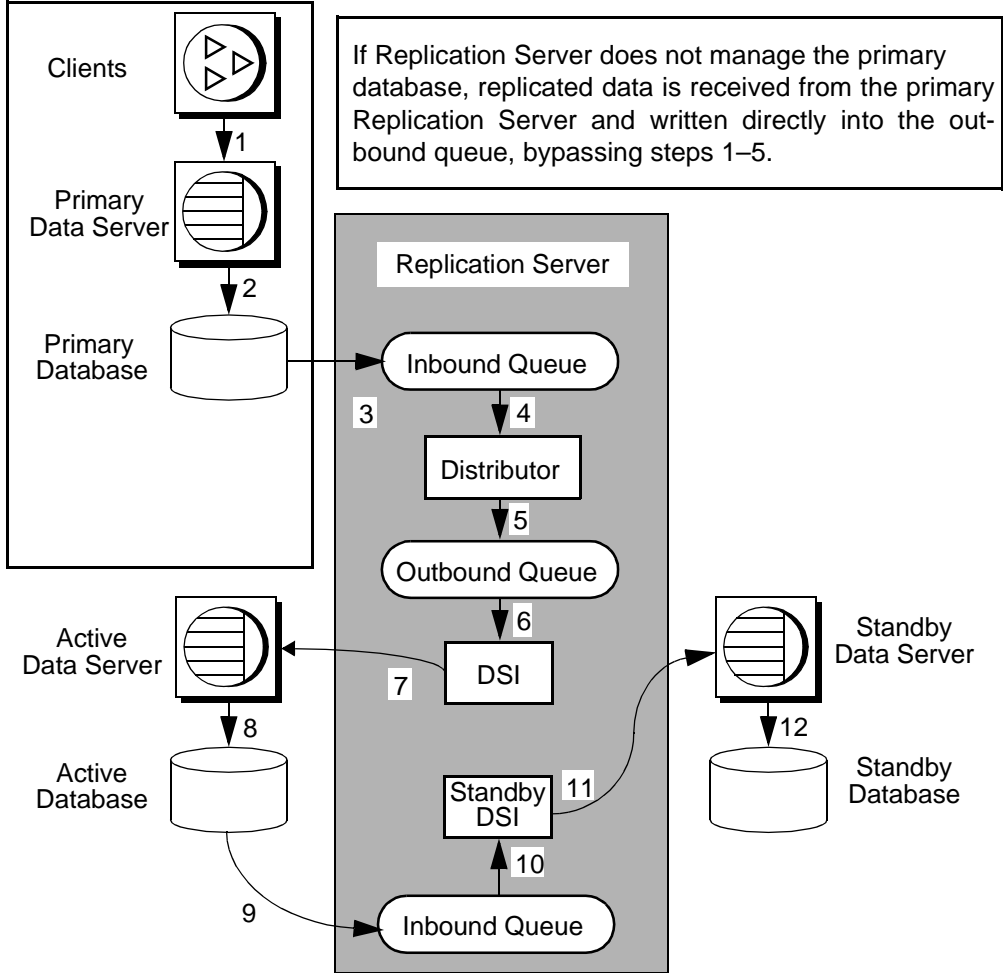
Figure 15-7 illustrates a warm standby application for a replicate database. In this example, a single Replication Server manages three databases:

- A primary database,
- The active database for a logical replicate database, and
- The standby database for a logical replicate database.

The logical replicate database has subscriptions for the data in the primary database. Therefore, updates from the primary database are replicated to both the active and the standby databases.

In this example, a single Replication Server manages both the primary and replicate databases. In other instances, different Replication Servers may manage the primary and replicate databases.

Figure 15-7: Warm standby application for a replicate database



The numbers in Figure 15-7 indicate the flow of transactions from client applications through the replication system in a warm standby application for a replicate database.

From client applications to primary and active databases

In Figure 15-7, numbers 1 through 8 trace transactions from clients to the primary database, and, via normal replication, to the active replicate database:

- Clients execute transactions in the primary data server.
- The primary data server updates the primary database.

- RepAgent for the primary database reads transactions for replicated data in the transaction log and forwards them to the Replication Server, which writes them into an inbound queue.
- The Distributor thread reads transactions from the inbound queue.
- The Distributor processes transactions against subscriptions and writes replicated transactions into an outbound queue.

If the Replication Server managing the warm standby application for the replicate database does not also manage the primary database, replicated data is received from the primary Replication Server and written directly to the outbound queue. Steps 1 through 5 are bypassed.

- A DSI thread reads transactions from the outbound queue.
- The DSI thread executes the transactions in the replicate data server, which is the active data server for the warm standby application.
- The active data server updates the active database.

If the transactions originate in a primary database managed by a different Replication Server, the Distributor thread in the primary Replication Server writes them into an RSI outbound queue. Then they are replicated to a DSI outbound queue in the replicate Replication Server in order to be applied in the active database for the logical replicate database.

From active database
to standby database

In Figure 15-7, numbers 9 through 12 trace transactions from the active database for the logical replicate database to its standby database:

- RepAgent for the active database reads the transactions in the active database log and forwards them to the Replication Server, which writes them into an inbound queue.

All transactions for replicated data, including those executed by the maintenance user, are sent to the Replication Server for application in the standby database.

- The standby DSI thread reads transactions from the inbound queue.
- The standby DSI thread executes transactions in the standby data server.
- The standby data server updates the standby database.

Configuring logical connection save intervals

This section describes some options for reconfiguring the save intervals for a logical replicate database. A save interval for a connection specifies how long messages will be retained in a stable queue before they can be deleted. If you set up a warm standby application through the usual procedure, the default settings will work correctly.

You can use the `configure logical connection` command to configure the DSI queue save interval and the materialization queue save interval for the logical connection.

Refer to “configure logical connection” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for the syntax of this command.

Warning! The DSI queue save interval and the materialization queue save interval settings for a logical connection should be reset *only* under serious conditions stemming from a lack of stable queue space. Resetting these save intervals (from `strict` to a given number of minutes) may lead to message loss at the standby database. Replication Server cannot detect this type of loss; you have to verify the integrity of the standby database yourself.

The DSI queue save interval

By default, the DSI queue save interval for the logical connection is set to `'strict'` when you create a standby database. This causes Replication Server to retain DSI queue messages until they are delivered to the standby database. If you must change the DSI queue save interval for the logical connection, use the `configure logical connection` command.

For example, to force a replicate Replication Server to save messages destined for its logical replicate data server LDS for one hour (sixty minutes), enter the following command:

```
configure logical connection to LDS.logical_pubs2
set save_interval to '60'
```

To reset this save interval back to `'strict'`, enter:

```
configure logical connection to LDS.logical_pubs2
set save_interval to 'strict'
```

The materialization queue save interval

The materialization queue save interval for the logical connection is set to `'strict'` by default when you create a subscription. This causes Replication Server to retain materialization queue messages until they are delivered to the standby database. If you must change the materialization queue save interval for the logical connection, use the `configure logical connection` command.

For example, to force a replicate Replication Server to save messages in the materialization queue for its logical replicate data server LDS for one hour (sixty minutes), enter the following command:

```
configure logical connection to LDS.logical_pubs2
set materialization_save_interval to '60'
```

To reset this save interval back to 'strict', enter:

```
configure logical connection to LDS.logical_pubs2
set materialization_save_interval to 'strict'
```

Using replication definitions and subscriptions

This section contains information about using warm standby databases with replication definitions and subscriptions. See “Warm standby applications using replication” on page 524 for more information about warm standby applications for a primary or replicate database.

Creating replication definitions for warm standby databases

Replication Server does not require replication definitions to maintain a standby database, although using replication definitions can improve performance when replicating into a standby database. You can create a replication definition for each table in the logical database. You can also use function replication definitions when replicating into a standby database.

Replication definitions can change how Replication Server replicates data into a standby database, allowing you to optimize your warm standby application or enable a non-default behavior that your application requires.

You can use replication definitions in a warm standby application in the following scenarios:

- To improve the performance of the replication system, as described under “Using replication definitions to optimize performance” on page 534.
- In normal replication into or out of the logical database, as described in “Warm standby applications using replication” on page 524.

alter table support for warm standby

Adaptive Server Enterprise version 12.0 and later allows users to alter existing tables— add non-nullable columns, drop columns, and modify column datatypes.

This section describes how Replication Server supports table changes resulting from the alter table command when the table has no subscriptions.

Note To support table changes that result from alter table when subscriptions exist for that table, you need to alter the table’s replication definition. See “Modifying replication definitions” on page 294 for instructions.

In previous releases, when a replication definition was defined for a table, Replication Server always used the column datatype defined in the warm standby replication definition. Beginning with Replication Server version 12.0, and depending on the situation, Replication Server may or may not use a table’s replication definition.

No replication definition

When you use alter table against a table without replication definitions, Replication Server sends warm standby databases the same information it receives from the primary server. All options of alter table are supported. When you execute alter table at the primary, the command is replicated to the warm standby, and replication to the standby continues—no action is required in the Replication Server.

See version 12.0 of the *Adaptive Server Enterprise Reference Manual, Volume2: Commands* for alter table syntax and information.

Alter table add column with default

When you issue the alter table command in the active database to add a column with a default value, Adaptive Server creates a constraint with an auto-generated name. When the command is replicated to the standby database, the standby database also creates the same constraint with another, different auto-generated name. When you drop the constraint in the active database, the standby database does not recognize the constraint name and generates a data server interface (DSI) error.

To avoid this, drop the constraint in the active database first. The data server interface (DSI) shuts down automatically. Then drop the constraint created in the standby database and issue the resume dsi skip transaction command.

An alternative workaround is to execute:

```
alter table <table name>
replace <column name>
default null
```

This automatically drops the constraints created on both active and standby sites.

Warm standby with no *send standby* clause

When there is no *send standby* clause associated with any replication definition, Replication Server sends whatever data it receives from the primary table without referring to the replication definitions.

Replication Server uses the original column names and datatypes to send data received from the RepAgent. The replication definition is used only to find the primary key. The primary keys are the union of primary keys in all replication definitions for the table.

If schema changes do not involve dropping all primary key columns in all replication definitions of the table, the scenario is the same as discussed in “No replication definition” on page 532. All options of *alter table* are supported, and no action is required in the Replication Server.

You can alter the replication definition at any point to drop all primary keys in the replication definitions, and add the new primary key columns to the replication definitions before you alter the primary table.

Drop the old primary keys only after all of the old data rows are out of the replication system. Otherwise, the Data Server Interface (DSI) shuts down. If this occurs, see for recovery instructions.

Warm standby with *send standby all columns* clause

When *send standby all columns* is associated with a replication definition, Replication Server sends whatever data it receives from the RepAgent using the original column names and datatypes. The replication definition is used only to find the primary key.

If schema changes do not involve dropping all primary key columns in the replication definition with the *send standby all columns* clause, the scenario is the same as “No replication definition” on page 532. All options of *alter table* are supported, and no action is required in the Replication Server.

You can alter the replication definition at any time to drop all primary keys in the replication definition with the `send standby all columns` clause, and add the new primary key columns to the replication definition before you alter the primary table.

Drop the old primary keys after all of the old data rows have left the replication system. Otherwise, the Data Server Interface (DSI) shuts down. If this occurs, see “Recovering from inbound queue problems” on page 299 for recovery instructions.

Warm standby with `send standby replication definition columns` clause

When there is a `send standby replication definition columns` clause in the replication definition, the standby will continue to use the replicate table name and column names as well as the datatype defined in the table’s corresponding replication definition.

If you want the replication definition datatype to be used in the standby, always create a replication definition with a `send standby replication definition columns` clause.

Please note that:

- To add or alter columns in the primary database, follow the “Migration procedure” on page 296.
- To drop columns in the primary database, you do not need to alter the replication definition of the table as long as you do not drop all primary key columns.
- To drop all primary key columns, alter the replication definition to add new primary key columns before you alter the primary table. You can drop the old primary keys when the old data rows have been removed from the replication system.

Using replication definitions to optimize performance

When you specify that you want to use a replication definition for replicating into a standby database:

- Replication Server optimizes updates and deletes by using the primary key defined in the replication definition to generate the `where` clause.

- You can specify whether Replication Server uses the replication definition's replicate minimal columns setting for replicating into the standby database. This setting indicates whether updates replace the values for all columns or only the columns with changed values.
- You can specify whether Replication Server replicates all of a table's columns or all of a stored procedure's parameters to the standby database or only those columns or parameters listed in the table or function replication definition.

Creating a replication definition for replicating into a standby database

To create a replication definition just for replicating into the standby database, use the send standby clause in the create replication definition command. The replication definition's primary key and replicate minimal columns setting will be used in replicating into the standby database.

Refer to “create replication definition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for detailed information about using this command.

Specifying a primary key

Replication Server generates a where clause to specify target rows for updates and deletes.

- If a replication definition for a table is marked with the send standby clause, the generated where clause contains only the columns listed in the primary key clause of the create replication definition command.
- If there are replication definitions for a table but none are marked with the send standby clause, the generated where clause contains the columns listed in the union of the primary key clauses of all of the replication definitions.
- If there is no replication definition for a table, the generated where clause includes all columns in the table except text, image, rawobject, rawobject in row, timestamp, and sensitivity columns.

Updating minimal columns

If you create a replication definition for replicating into a standby database, you can take advantage of another replication system performance optimization, the minimal columns setting.

When you use the replicate minimal columns clause, replicated update and delete transactions include only the required columns. Values for unchanged columns can be omitted from update commands. Omitting the unnecessary columns reduces the size of messages delivered through the replication system and requires Adaptive Server to do less work.

If you are not using replication definitions for replicating into the standby, you can still attain this performance benefit.

Minimal column replication occurs automatically if you have no replication definitions for a table or if you have replication definitions for a table but do not use one for replicating into the standby database.

Specifying columns to replicate into the standby database

If you create a replication definition for replicating into a standby database, you can specify which set of columns to replicate:

- Specify `send standby` or `send standby all columns` to replicate all the columns in the table into the standby database.
- Specify `send standby replication definition columns` to replicate only the replication definition's columns into the standby database.

Refer to “create replication definition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information about using the `send standby` clause with the `create replication definition` command.

Specifying parameters to replicate into the standby database

If you create a function replication definition, you can specify which set of parameters to replicate:

- Specify `send standby all parameters` (or omit the `all parameters` clause) to replicate all the parameters for the stored procedure into the standby database.
- Specify `send standby replication definition parameters` to replicate only the replication definition's parameters into the standby database.

If a replicated stored procedure has no function replication definition, when the stored procedure is executed, Replication Server replicates all of its parameters from the active database into the standby database. You can create only one function replication definition per replicated stored procedure.

Refer to “create function replication definition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information about using the `send standby` clause with the `create function replication definition` command.

Using replication definitions for tables with more than 1024 columns

Adaptive Server limits the number of expressions in the `where` clause to 1024. For warm standby applications, you must use replication definitions to replicate tables with more than 1024 columns, and make sure that the primary key is not more than 1024 columns. Adaptive Server generates an error if the Replication Server `where` clause has more than 1024 columns.

See “Using replication definitions to optimize performance” on page 534 for more information about the primary key and replication into the standby database.

Using replication definitions to copy redundant updates

Without a replication definition, Replication Server does not replicate redundant updates to the warm standby. That is, if an update merely changes the current value to the same value, and thus the before and after images are identical, Replication Server does not replicate the update.

However, if you want to replicate redundant updates, create a replication definition for the column that includes the `send standby replication definition parameters` option.

If you create a replication definition for a column, Replication Server *always* sends redundant updates, even when the replication definition is created with the `replicate minimal columns` option.

Using subscriptions with warm standby application

Although subscriptions are not used in replicating from the active to the standby database, you can:

- Create subscriptions for the data in a logical primary database, or
- Create subscriptions in order to replicate data from other databases into a logical replicate database.

The `create subscription` and `define subscription` commands use the logical database and data server names instead of the physical names.

See “Warm standby applications using replication” on page 524 for more information about warm standby applications for a primary or replicate database. Also see Chapter 11, “Managing Subscriptions” for more information about subscriptions and subscription materialization.

Restrictions on using subscriptions

Replication Server supports all forms of subscription materialization and dematerialization in warm standby applications. These restrictions apply to the creation of subscriptions that replicate data from or into warm standby databases:

- When there is a logical connection for a database, you cannot create a subscription for the physical active or standby database. You must create the subscription for the logical database in order to replicate subscription data into or from both the active and standby databases.
- You cannot create subscriptions while adding the standby database to the replication system. You must wait until the standby database has been properly initialized.
- You cannot add the standby database to the replication system while any subscriptions are being created.
- You cannot create new subscriptions while the switch active command is executing.

Subscription materialization for logical primary database

This section describes subscription materialization issues for a logical primary database. It also describes what happens if you execute the switch active command for a logical primary database during subscription materialization.

During subscription materialization, data is selected from the active primary database into a materialization queue.

When you execute the switch active command, the primary Replication Server replicates RSSD information to notify replicate sites that the active database has been changed. When a replicate Replication Server with a materializing subscription receives this information, the materialization queue is dropped. A new queue is built by reselecting the subscription data from the new active primary database.

Note The RepAgent thread for the RSSD of the primary Replication Server must be running for replicate Replication Servers to detect that the active database has been changed.

Subscription materialization for logical replicate database

This section describes subscription materialization issues for a logical replicate database. It also describes what happens if you execute the switch active command for a logical replicate database during subscription materialization.

The following sections discuss each subscription materialization method.

Atomic materialization When you use atomic materialization, Replication Server sets the save interval for the materialization queue to 'strict'. Transactions are not deleted from the materialization queue until the data has been applied to the active database and replicated into the standby database.

Replication Server executes a marker in the active replicate database when the materialization queue has been applied. The marker marks the start of transactions that execute after the materialization queue is applied.

When the marker is executed at the active replicate database, Replication Server writes an informational message like this in its log:

```
I. 95/10/03 18:00:15. REPLICATE RS: Created atomic subscription
<publishers_sub> for replication definition <publishers_rep> at active
replicate for <LDS.pubs2>
```

When the marker arrives at the standby replicate database, Replication Server writes an informational message like this in its log:

```
I. 95/10/03 18:00:15. REPLICATE RS: Created atomic subscription
<publishers_sub> for replication definition <publishers_rep> at standby
replicate for <LDS.pubs2>
```

Materialization is now complete and Replication Server drops the materialization queue. The subscription is considered **VALID** at both the active and the standby replicate database.

If you execute the `switch active` command while the materialization queue is being processed, Replication Server reapplies the materialization queue to the new active database. If you used the `incrementally` option to create the subscription, only the batches of materialization rows that were not already replicated into the new active database are reexecuted.

Nonatomic materialization

When you use nonatomic materialization, the save interval is set to 0, allowing Replication Server to delete rows from the materialization queue after they are applied to the active database.

If a subscription is materializing when you execute the `switch active` command, Replication Server finishes processing the materialization queue, but marks the subscription "suspect." Use the `check subscription` command to find the subscription status in the active and replicate databases. You must drop and recreate suspect subscriptions.

Bulk materialization

If you use bulk materialization to create a subscription that replicates data into a warm standby application, you must ensure that the subscription data is loaded into the active and standby replicate databases.

If you load the data with a method that logs the inserted rows, such as logged bcp, Replication Server replicates the rows into the standby database. If you load the data with a non-logged method, you must also load it into the standby database because the active database log contains no insert records to replicate into the standby database.

During bulk materialization, you execute the activate subscription with suspension command before you load the subscription data into the replicate database. By default, activate subscription with suspension suspends the DSI threads for both the active database and the standby database. Suspending DSI threads allows you to load the data into both databases.

If you load the data using logged bcp or some other method that logs the rows, execute activate subscription with suspension at active replicate only so that Replication Server only suspends the DSI for the active database. This allows the inserted rows to be replicated from the active database into the standby database.

Checking subscriptions

For a warm standby application for a logical replicate database, you can use the check subscription command to check subscription status. The Replication Server managing the warm standby application returns either one or two status messages, depending on whether or not the status is different for the active and the standby database.

For example, while you are creating a subscription, the materialization status may be VALID at the active database and ACTIVATING at the standby database.

Dropping subscriptions

For a logical replicate database, you can drop a subscription using the drop subscription command with the with purge option. A drop subscription marker follows the dematerialization data from the DSI queue to the active database, and then travels to the standby database. After the marker has been received at both databases, subscription data is deleted from both databases.

While executing
switch active

You can execute the switch active command at the replicate Replication Server while you drop a subscription using the drop subscription command with the with purge option. Replication Server suspends DSI threads and temporarily suspends dematerialization. After switch active completes, the DSI threads are resumed and dematerialization restarts.

**Suspect drop
subscription**

Dropping a subscription using the `with purge` option for a logical replicate database may lead to a suspect drop subscription if:

- The subscription is materializing in the active database, and
- You switch the active and standby databases, then
- You drop the subscription while it is materializing in the new active database.

Dematerialization restarts and proceeds normally for the new active database, but the new standby (old active) database may retain some subscription data that is not purged. To resolve the discrepancy, you can reconcile the active and the standby database using the `rs_subcmp` program, or you can drop and re-create the standby database.

For example, you may see a warning message like this when you try to execute `drop subscription`:

```
W. 95/10/02 20:59:15. WARNING #28171 DSI(111 SYDNEY_DS.pubs2) -  
/sub_dsi.c(1231)  
REPLICATE RS: Dropped subscription <publishers_sub> for replication  
definition <publishers_rep> at standby replicate for <SYDNEY_DS.pubs2>  
before it completed materialization at the Active Replicate. Standby  
replicate may have some subscription data rows left in the database
```

Missing columns when you create the standby database

When you create a standby database for an existing database that has replication definitions, missing columns may result under the following combination of circumstances:

- If the existing database has a replication definition that does not include all columns in the table, and
- An insert or update transaction that has not been committed is in the inbound queue, and
- You create a standby database for the existing database (now the active database), after which
- The transaction commits.

Although, by default, a standby database is supposed to receive all columns, at the time the transaction began, the standby database did not exist. Replication Server would have discarded values for columns not in the replication definition. If a column is not in the replication definition and the standby database allows a null value for the column, the row can be inserted into or updated in the standby database without the missing value. Otherwise, you must reconcile the databases yourself.

Loss detection and recovery

Creating a warm standby application introduces additional types of loss detection messages into a replication system. See Chapter 19, “Replication System Recovery” for general information on Replication Server recovery, and for recovery procedures.

If you rebuild queues in a Replication Server that participates in a warm standby application, the Replication Server may detect losses between any of the following databases:

Table 15-8: Loss detection in warm standby applications

Loss detected from	To
Logical replicate database	Logical primary database
Logical primary database	Physical replicate database
Physical primary database	Logical replicate database
Physical active database	Physical standby database
Logical primary database	Replication Server

If you need to use the ignore loss command in database recovery operations where a warm standby application is involved, use the same logical or physical data server and database designations that appear in the loss detection messages you received.

Performance Tuning

To meet the needs and demands of your Replication Server system, you must manage resources effectively and optimize the performance of individual Replication Servers. You can affect the performance of a Replication Server by changing the values of configuration parameters, by using parallel DSI threads, or by choosing disk allocations. To manage these resources successfully, you should understand something about Replication Server's internal processing.

Name	Page
Replication Server internal processing	543
Configuration parameters that affect performance	551
Suggestions for using tuning parameters	559
Using parallel DSI threads	566
Using multiprocessor platforms	592
Allocating queue segments	593

Replication Server internal processing

During replication, data operations are carried out by several Replication Server **threads**. On UNIX platforms, they are POSIX threads. On Windows platforms, they are WIN32 threads. Replication Server also stores data in queues and relies on the Replication Server System Database (RSSD) for critical system information. This section describes how these internal operations support various processes within the primary and replicate Replication Servers.

Threads, modules, and daemons

Replication Server runs multiple threads concurrently. The total number of threads depends on the number of databases that a Replication Server manages and the number of Replication Servers to which it has direct routes. Each thread performs a specific function such as managing a user session, receiving messages from a RepAgent, receiving messages from another Replication Server, or applying transactions to databases.

Some threads call specific portions (or “**modules**”) of Replication Server to determine the destination of messages and transactions, and to determine what operations to replicate and how to replicate them.

Daemon threads, which run in the background and perform specified operations at predefined times or in response to certain events, run during such Replication Server activities as subscription materialization.

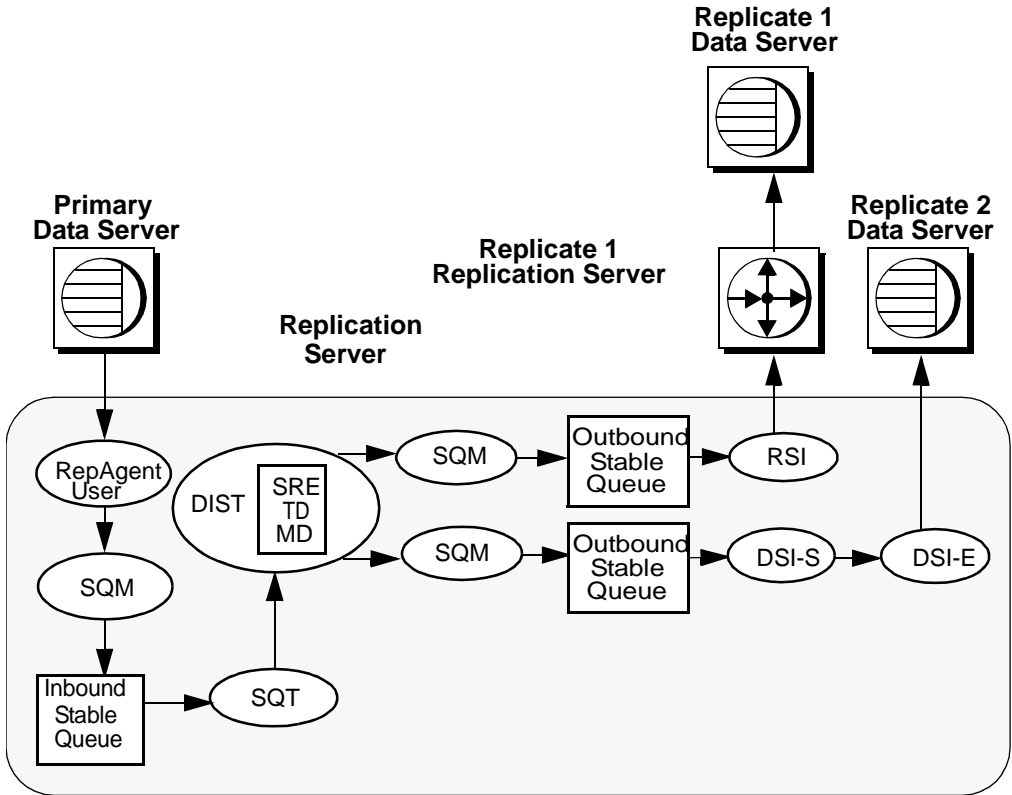
For details on Replication Server threads, modules, and daemons involved in processes specific to the primary Replication Server, see “Processing in the primary Replication Server” on page 544.

When you troubleshoot the replication system, verify the status of Replication Server threads, modules, and daemons. See Chapter 13, “Verifying and Monitoring Replication Server” for details.

Processing in the primary Replication Server

This section describes how a transaction that originates in a primary data server is sent to the primary Replication Server and subsequently distributed to a replicate Replication Server as illustrated in Figure 16-1.

Figure 16-1: Threads used for processing in the primary Replication Server



Replication agent user thread

The information in this section applies to all replication agents.

RepAgent logs in to Replication Server through an Open Client interface. It scans the transaction log, converts log records directly into LTL (Log Transfer Language) commands, and sends them to Replication Server as soon as they are logged—either in batches or one at a time. Replication Server then distributes the transaction information to subscribing replicate databases.

Replication Server has one RepAgent user thread for each primary database that it manages. Thus, Replication Server has one RepAgent user thread for each RepAgent. The RepAgent user thread verifies that RepAgent submissions are valid and writes them into the inbound stable queue for the database.

Stable Queue Manager thread

There is one Stable Queue Manager (SQM) thread for each stable queue accessed by the primary Replication Server, whether inbound or outbound. Each RepAgent user thread works with a dedicated SQM thread that reclaims stable queue space after a transaction is forwarded to a data server or to another Replication Server.

Stable Queue Transaction thread

Commands stored in transaction log records and in the inbound queue are ordered according to the sequence in which they were committed—although they are not necessarily grouped by transaction. It is the task of the Stable Queue Transaction (SQT) thread to reassemble transactions and place the transactions in commit order. Transactions must be in commit order for final application on the destination's data servers and for materialization processing.

The SQT thread reassembles transactions as it reads commands from its stable queue and keeps a linked list of transactions. When it reads a commit record, the SQT thread makes that transaction available to the distributor thread or to the DSI thread, depending on what process required the SQT ordering of the transaction.

When it reads a rollback record, the SQT thread tells the SQM thread to delete affected records from all stable queues. The SQT thread also notifies the DSI thread when a transaction exceeds the large transaction threshold. See “Using parallel DSI threads” on page 566 for more information on transaction thresholds.

Distributor thread and related modules

For each primary database managed by a Replication Server, there is a distributor (DIST) thread, which in turn uses SQT to read from the inbound queue and SQM threads to write transactions to the outbound queue. Thus, for example, if there are three primary databases, then there are three inbound queues, and three DIST and SQT threads.

In determining the destination of each transaction row, the DIST thread makes calls to the following modules: Subscription Resolution Engine, Transaction Delivery, and Message Delivery. All DIST threads share these modules. These modules, and the role they play in the replication system, are described in the following sections.

Subscription resolution engine

The subscription resolution engine (SRE) matches transaction rows with subscriptions. When it finds a match, it attaches a destination-database ID to each row. It marks only rows required for subscriptions, thereby minimizing network traffic. If no subscriptions match, the DIST thread discards the row data.

For each row, the SRE determines whether subscription migration occurs.

- A row migrates *into* a subscription when its column values change so that the row matches the subscription and must be added to the replicate table.
- A row migrates *out of* a subscription when its column values change so that it no longer matches the subscription and must be deleted from the replicate table.

When the SRE detects subscription migration, it determines which operation to replicate (insert, delete, or update) to maintain consistency between the replicate and primary tables.

Transaction Delivery module

The Transaction Delivery (TD) module is called by the DIST thread to package transaction rows for distribution to data servers and other Replication Servers.

Message Delivery module

The Message Delivery (MD) module is called by the DIST thread to optimize routing of transactions to data servers or other Replication Servers. The DIST thread passes the transaction row and the destination ID to the MD module. Using this information and routing information in the RSSD, the module determines where to send the transaction:

- To a data server via a DSI thread, or
- To a Replication Server via an RSI thread.

After determining how to send the transaction, the MD module places the transaction into the appropriate outbound queue.

Data Server Interface threads

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

Each DSI thread is composed of a scheduler thread (DSI-S) and one or more executor threads (DSI-E). Each DSI executor thread opens an Open Client connection to a database.

To improve performance in sending transactions from a Replication Server to a replicate database it manages, you can configure a database connection so that transactions are applied using more than one DSI executor thread. See “Using parallel DSI threads” on page 566 for a description of this feature.

The DSI scheduler thread calls the SQT interface to:

- Collect small transactions into groups by commit order
- Dispatch transaction groups to the next available DSI executor thread

The DSI executor threads:

- Map functions using the function strings defined for the functions, according to the function-string class assigned to the database connection
- Execute the transactions in the replicate database
- Take action on any errors returned by the data server; depending on the assigned error actions, also record any failed transactions in the exceptions log

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 replicate data server.

Replication Server Interface thread

RSI threads are asynchronous interfaces to send messages from one Replication Server to another. One RSI thread exists for each destination Replication Server to which the source database has a direct route.

The DIST thread in 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 replicate Replication Server and transfers messages from the stable queue to the replicate 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 replicate Replication Server. When an indirect route is created, Replication Server does not create a new stable queue and RSI thread. Instead, messages for indirect routes are handled by the RSI thread for the direct route. For details, see Chapter 6, “Managing Routes.”

Miscellaneous daemon threads

The Replication Server daemon threads shown in Table 16-1 perform miscellaneous tasks in the replication system.

Table 16-1: Additional Replication Server daemon threads

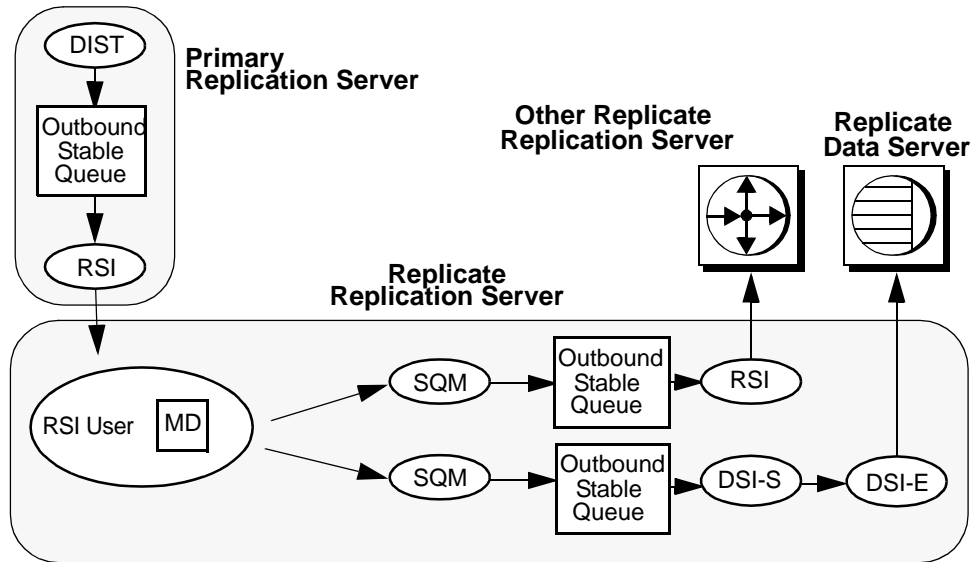
Thread or daemon name	Description
Alarm daemon (dALARM)	The alarm daemon keeps track of alarms set by other threads, such as the fade-out time for connections and the interval for the subscription retry daemon.
Asynchronous I/O daemon (dAIO)	The asynchronous I/O daemon manages asynchronous I/O to Replication Server stable queues.
Connection manager daemon (dCM)	The connection manager daemon manages connections to data servers and other Replication Servers.
Recovery daemon (dREC)	The recovery daemon takes care of various operations in connection with warm standby applications, routing, and recovery procedures.
Counter daemon (dSTATS)	The counter daemon manages the interface for flushing counters to the RSSD, and calculates derived values when it awakens.
Subscription retry daemon (dSUB)	The subscription retry daemon wakes up after a configurable timeout period (<code>sub_daemon_sleep_time</code> configuration parameter in the <code>rs_config</code> system table) and attempts to resume processing for subscriptions that may have failed.
Version daemon (dVERSION)	The version daemon activates briefly when the Replication Server is started for the first time after an upgrade. It communicates the Replication Server new version number to the ID Server.
RS user thread	The RS user thread manages connections from replicate Replication Servers during the process of creating or dropping subscriptions. See “Subscription materialization methods” on page 351 for the data flow involved in creating and dropping subscriptions.
USER thread	A USER thread is created when a user logs in to a Replication Server to execute RCL commands.

Processing in the replicate Replication Server

This section describes the processes involved when a replicate Replication Server receives incoming messages from a primary Replication Server.

“Processing in the primary Replication Server” on page 544 describes processing for some of the threads—SQM, RSI, DSI—described in this section. Refer to Figure 16-1 on page 545.

Figure 16-2: Transaction processing in the replicate Replication Server



RSI user thread

The RSI user thread is a client connection thread for incoming messages from another Replication Server. It calls the Message Delivery (MD) module to determine whether to send the message to:

- A data server using the DSI thread, described in “Data Server Interface threads” on page 547. The DSI thread is composed of a scheduler thread (DSI-S) and one or more executor threads (DSI-E).
- Another Replication Server using the RSI thread, described in “Replication Server Interface thread” on page 548.

The RSI user thread writes commands destined for other Replication Servers or databases into outbound queues. See “Processing in the primary Replication Server” on page 544 for details on how messages are processed after they are stored in the outbound queues.

Configuration parameters that affect performance

Replication Server provides configuration parameters for improving performance that affect the entire server, or are targeted for individual connections or routes.

Replication Server parameters that affect performance

`rs_init` sets default configuration parameters after you install your Replication Server. You can change the values of the configuration parameters shown in Table 16-2 to improve Replication Server performance.

See “Changing Replication Server parameters” on page 102 for information on how to modify these parameters using `configure replication server`.

Table 16-2: Replication Server parameters that affect performance

Configuration parameter	Description
<code>exec_cmds_per_timeslice</code>	<p>Specifies the number of LTL commands an LTI or RepAgent executor thread can process before yielding the CPU. By increasing this value, you allow the RepAgent Executor to control CPU resources for longer periods of time, which may improve throughput from RepAgent to Replication Server.</p> <p>See “Controlling the number of commands the RepAgent executor can process” on page 562.</p> <p>Default: 5</p> <p>Minimum: 1</p> <p>Maximum: 2,147,483,647</p>
<code>exec_sqm_write_request_limit</code>	<p>Specifies the amount of memory available to the LTI or RepAgent executor thread to queue up outstanding write requests. Once the limit is reached, the RepAgent Executor thread sleeps until some of the outstanding writes have been completed.</p> <p>See “Controlling the number of outstanding bytes” on page 562.</p> <p>Default: 16,384 bytes</p> <p>Minimum: 16,384</p> <p>Maximum: 983,040 bytes</p>

Configuration parameter	Description
<code>init_sqm_writer_delay</code>	<p>The initial amount of time an SQM Writer should wait for more messages before writing a partially full block of messages to the queue. The SQM Writer always tries to write full blocks to the queue. If it has partially filled a block, and cannot fill it, SQM Writer waits the amount of time specified by <code>init_sqm_writer_delay</code> before rechecking whether messages are waiting to be added to the block. If no messages exist, SQM Writer doubles the <code>init_sqm_write_delay</code> time. The SQM Writer continues to double the delay time until it reaches the value of <code>init_sqm_writer_max_delay</code>. At this point, SQM Writer writes the partially full block.</p> <p>See “Setting the amount of time SQM Writer waits” on page 559.</p> <p>Default: 1000 milliseconds</p>
<code>init_sqm_writer_max_delay</code>	<p>The maximum amount of time an SQM Writer thread should wait for more messages before writing a partially full block of messages to the queue. See the description of <code>init_sqm_write_delay</code> for more information. See also “Setting the amount of time SQM Writer waits” on page 559.</p> <p>Default: 10,000 milliseconds</p>
<code>md_sqm_write_request_limit</code>	<p>Specifies the amount of memory available to the Distributor thread to queue up outstanding write requests. Once the limit is reached, the Distributor thread sleeps until some of the outstanding writes have been completed. See “Controlling the number of outstanding bytes” on page 562.</p> <p>Default: 100,000 bytes</p> <p>Minimum: 65,536 bytes</p> <p>Maximum: 983,040 bytes</p>
<code>memory_limit</code>	<p>The maximum total memory the Replication Server can use.</p> <p>Values for several other configuration parameters are directly related to the amount of memory available from the memory pool indicated by <code>memory_limit</code>. These include <code>exec_sqm_write_request_limit</code>, <code>md_sqm_write_request_limit</code>, <code>queue_dump_buffer_size</code>, <code>sqm_max_cache_size</code>, <code>sre_reserve</code>, and <code>sts_cachesize</code>.</p> <p>Default: 20MB</p>
<code>rec_daemon_sleep_time</code>	<p>Specifies the sleep time for the recovery daemon, which handles “strict” save interval messages in warm standby applications and certain other operations. See “Setting wake up intervals” on page 560.</p> <p>Default: 2 minutes</p>
<code>send_enc_password</code>	<p>Ensures that all Replication Server client connections are made with encrypted passwords—except for the first connection to the RSSD. Values are “on” and “off.”</p> <p>See “Sending encrypted passwords for Replication Server client connections” on page 202 for more information.</p> <p>Default: off</p>

Configuration parameter	Description
<code>smp_enable</code>	<p>Enables symmetric multiprocessing (SMP). Specifies whether Replication Server threads should be scheduled internally by Replication Server or externally by the operation system. When Replication Server threads are scheduled internally, Replication Server is restricted to one machine processor, regardless of how many may be available. Values are “on” and “off.”</p> <p>See “Making SMP more effective” on page 563.</p> <p>Default: off</p>
<code>sqm_recover_segs</code>	<p>Specifies the number of stable queue segments Replication Server allocates before updating the RSSD with recovery QID information.</p> <p>See “Specifying the number of stable queue segments allocated” on page 563.</p> <p>Default: 1 Minimum: 1 Maximum: 2,147,483,648</p>
<code>sqm_write_flush</code>	<p>Specifies whether or not writes to memory buffers are flushed to the disk before the write operation completes. Values are “on” and “off.”</p> <p>See also “Stable devices: considerations.”</p> <p>Default: on</p>
<code>sqt_init_read_delay</code>	<p>The length of time an SQT thread sleeps while waiting for an SQM read before checking to see if it has been given new instructions in its command queue. With each expiration, if the command queue is empty, SQT doubles its sleep time up to the value set for <code>sqt_max_read_delay</code>.</p> <p>Default: 2000 milliseconds (ms) Minimum: 1000 ms Maximum: 86,400,000 ms (24 hours)</p>
<code>sqt_max_cache_size</code>	<p>Maximum SQT cache memory, in bytes. See “Sizing the SQT cache” on page 560.</p> <p>Default: 1,048,576 bytes</p>
<code>sqt_max_read_delay</code>	<p>The maximum length of time an SQT thread sleeps while waiting for an SQM read before checking to see if it has been given new instructions in its command queue.</p> <p>Default: 10,000 ms (10 seconds) Minimum: 1000 ms Maximum: 86,400,000 ms (24 hours)</p>
<code>stats_daemon_sleep_time</code>	<p>Number of seconds between RSSD flushes. See “Setting wake up intervals” on page 560.</p> <p>Default: 600 seconds Minimum: 10 seconds Maximum: 1,814,400 seconds (3 weeks)</p>

Configuration parameter	Description
<code>sts_cachesize</code>	<p>The total number of rows that are cached for each cached RSSD system table. Increasing this number to the number of active replication definitions prevents Replication Server from executing expensive table lookups.</p> <p>Monitor whether the STS cache is too small by reviewing counter 11008 – <code>STSCacheExceed</code> or examining the Replication Server log for warnings that rows have been removed from the STS cache. See “Caching system tables” on page 560.</p> <p>Default: 100</p>
<code>sts_full_cache_table_name</code>	<p>Specifies an RSSD system table that is to be fully cached. Fully cached tables do not require access to the RSSD for simple select statements.</p> <p>See “Caching system tables” on page 560 for a list of RSSD tables that can be fully cached.</p>
<code>sub_daemon_sleep_time</code>	<p>Number of seconds the subscription daemon sleeps before waking up to recover subscriptions. The range is 1 to 31,536,000.</p> <p>See “Setting wake up intervals” on page 560.</p> <p>Default: 120 seconds</p>
<code>sub_sqm_write_request_limit</code>	<p>Specifies the memory available to the subscription materialization/dematerialization thread. Once the limit is reached, the subscription management thread blocks until enough data is written to the queue to bring the amount of outstanding bytes to be written under this threshold.</p> <p>Default: 16,384 bytes</p> <p>Minimum: 16,384 bytes</p> <p>Maximum: 983,040</p>

Stable devices: considerations

Like any application, Replication Server is subject to standard I/O and I/O device best practices. You should consider the impact of contention for disk Read/Write heads and I/O channels when planning how your stable devices will be used to support your stable queues. To the extent that you can dedicate one or more devices to each queue, I/O will be less of a performance issue. This includes guarding the devices from use by other processes such as primary or replicate databases or RSSDs. You can use the database connection parameter `disk_affinity` to establish affinities between queues and specific partitions that are supported by dedicated devices.

For stable queues initialized on UNIX operating system files, the `sqm_write_flush` configuration parameter controls whether or not writes to memory buffers are flushed to the disk before the write operation completes.

When `sqm_write_flush` is on, Replication Server opens stable queues using the `O_DSYNC` flag. This flag ensures that writes are flushed from memory buffers to the disk before write operations complete. Because the data is stored on physical media, Replication Server can always recover the data in the event of a system failure. This is the default setting.

When `sqm_write_flush` is off, writes may be buffered in the UNIX file system. If subsequent writes fail, automatic recovery is not guaranteed. Testing has shown that when comparing the write rates of the various options for partition types and I/O flushing that writing to a buffered file system with `sqm_write_flush` on is up to five times slower than writes to raw partitions. Further, testing has shown that writes to raw partitions are up to seven times slower than writes to buffered file systems with `sqm_write_flush` off. Turning `sqm_write_flush` off when using UNIX Buffered file systems for stable devices provides peak I/O performance but with an increased risk of data loss. Provided you keep primary database transaction log backups, that risk can be reduced or eliminated.

Note The `sqm_write_flush` setting is ignored for stable queues initialized on raw partitions or Windows files. In these cases, write operations always take place directly to media.

Connection parameters that affect performance

Table 16-3 describes the database connection parameters that can affect performance. See Chapter 7, “Managing Database Connections,” for a complete list of connection parameters.

Table 16-3: Connection parameters that affect performance

Configuration parameter	Description
<code>batch</code>	The default, “on,” allows command batches to a replicate database. Default: on
<code>db_packet_size</code>	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 a System 10 or later SQL Server or Adaptive Server that has been reconfigured. Default: 512-byte network packet for all Adaptive Server databases

Configuration parameter	Description
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
dsi_cmd_batch_size	The maximum number of bytes that Replication Server places into a command batch. Default: 8192 bytes
dsi_commit_check_locks_intrvl	The number of milliseconds (ms) the DSI executor thread waits between executions of the <code>rs_dsi_check_thread_lock</code> function string. Used with parallel DSI. See “Using parallel DSI threads” on page 566. Default: 1000 ms (1 second) Minimum: 0 Maximum: 86,400,000 ms (24 hours)
dsi_commit_check_locks_max	The maximum number of times the DSI executor thread executes the <code>rs_dsi_check_thread_lock</code> function string before rolling back and retrying a transaction. Used with parallel DSI. See “Using parallel DSI threads” on page 566. Default: 400 Minimum: 1 Maximum: 1,000,000
dsi_commit_control	Specifies whether commit control processing is handled internally by Replication Server using internal tables (on) or externally using the <code>rs_threads</code> system table (off). Used with parallel DSI. See “Using parallel DSI threads” on page 566. Default: on
dsi_large_xact_size	The number of commands allowed in a transaction before the transaction is considered to be large. Minimum: 4 Default: 100
dsi_max_xacts_in_group	Specifies the maximum number of transactions in a group. Larger numbers may reduce commit processing at the replicate database, and thereby improve throughput. Range of values: 1 – 100. See “Specifying the number of transactions in a group” on page 564. Default: 20
dsi_num_large_xact_threads	The number of parallel DSI threads to be reserved for use with large transactions. The maximum value is one less than the value of <code>dsi_num_threads</code> . Default: 0
dsi_num_threads	The number of parallel DSI threads to be used. The maximum value is 255. Default: 1

Configuration parameter	Description
<code>dsi_partitioning_rule</code>	<p>Specifies the partitioning rules (one or more) the DSI uses to partition transactions among available parallel DSI threads. Values are <code>origin</code>, <code>origin_sessid</code>, <code>none</code>, <code>time</code>, <code>user</code>, and <code>name</code>. See also “Partitioning rules: reducing contention and increasing parallelism” on page 576.</p> <p>You can specify more than one value at a time. Separate values with a comma, but no spaces. For example:</p> <pre>alter connection to TOKYO_DS.pubs3 set dsi_partitioning_rule to 'time,user'</pre> <p>Default: <code>none</code></p>
<code>dsi_serialization_method</code>	<p>Specifies the method used to maintain serial consistency between parallel DSI threads when applying transactions to a replicate data server.</p> <ul style="list-style-type: none"> <code>no_wait</code> – specifies that a transaction can start as soon as it is ready—without regard to the state of other transactions. <code>wait_for_start</code> – specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started. <code>isolation_level_3</code> – specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started, and that transaction isolation level 3 locking be used in the replicate data server. <code>wait_for_commit</code> – specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it is ready to commit. <code>none</code> – same as <code>wait_for_start</code>. Retained for backward compatibility. <code>single_transaction_per_origin</code> – same as <code>wait_for_start</code> with <code>dsi_partitioning_rule</code> set to <code>origin</code>. Retained for backward compatibility. <p>Default: <code>wait_for_commit</code></p>
<code>dsi_sqt_max_cache_size</code>	<p>Maximum SQT (Stable Queue Transaction) interface cache memory for the database connection, in bytes.</p> <p>The default, 0, means the current setting of the <code>sqt_max_cache_size</code> parameter is used as the maximum cache size for the connection.</p> <p>Default: 0</p>
<code>dsi_xact_group_size</code>	<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. <code>-1</code> means no grouping.</p> <p>Sybase recommends that you set <code>dsi_xact_group_size</code> to the maximum value and use <code>dsi_max_xacts_in_group</code> to control the number of transactions in a group.</p> <p>Maximum: 2,147,483,647</p> <p>Default: 65,536 bytes</p>
<code>exec_cmds_per_timeslice</code>	<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 2,147,483,648.</p> <p>Default: 5</p>

Configuration parameter	Description
exec_sqm_write_request_limit	Specifies the amount of memory available to the LTI or RepAgent Executor thread for messages waiting to be written to the inbound queue. Default: 16,384 bytes Minimum: 16,384 bytes Maximum: 983,040 bytes
md_sqm_write_request_limit	Specifies the amount of memory available to the Distributor for messages waiting to be written to the outbound queue. The range is 65,536 to 983,040 bytes. Default: 100,000 bytes
parallel_dsi	A shorthand method for configuring parallel DSI to default values. A value of “on” sets dsi_num_threads to 5, dsi_num_large_xact_threads to 2, dsi_serialization_method to wait_for_commit, and dsi_sqt_max_cache_size to 1 million bytes. A value of “off” sets the parallel DSI values to their defaults. You can set this parameter to “on” and then set individual parallel DSI configuration parameters to fine-tune your configuration. Default: off

Route parameters that affect performance

Table 16-4 describes the route configuration parameters that affect performance. See Chapter 6, “Managing Routes,” for a complete list of route parameters.

Table 16-4: Route parameters that affect performance

Configuration parameter	Description
rsi_batch_size	The number of bytes sent to another Replication Server before a truncation point is requested, which allows Replication Server to delete messages in the source RSI queue. The range is 1024 to 262,144. Default: 262,144 bytes
rsi_packet_size	Packet size, in bytes, for communications with other Replication Servers. The range is 1024 to 8192. Default: 2048 bytes
rsi_sync_interval	The number of seconds between RSI synchronization inquiry messages. The Replication Server uses these messages to synchronize the RSI outbound queue with destination Replication Servers. The value must be greater than 0. Default: 60 seconds

Suggestions for using tuning parameters

This section provides basic recommendations for improving Replication Server performance. Whether or not changing these configuration values improves your system performance depends on your system configuration and how Replication Server is used at your site.

Setting the amount of time SQM Writer waits

Replication Server configuration parameters: `init_sqm_write_delay` and `init_sqm_write_max_delay`

In a low-volume system, set `init_sqm_write_delay` and `init_sqm_write_max_delay` to a low value so that the SQM Writer need not wait long before writing a partially full block. In a high-volume system, set these parameters higher because the SQM Writer rarely waits to fill a block.

Monitor how often the SQM Writer waits by reviewing counter 6038 – `WritesTimerPop`.

Determine the number of full or partially full blocks that have been written by reviewing these counters:

- 6002 – `BlocksWritten`
- 6041 – `BlocksFullWrite`

If counter 62006 – `SleepsWriteQ` is relatively high compared to counter 62002 – `BlocksRead`, SQM Readers must too often wait for the next block of messages to deliver downstream—which causes latency. Decrease the values of `init_sqm_writer_delay` and `init_sqm_writer_max_delay` so that SQM Writer does not wait to long before writing a partially full block.

Ideally, the ratio of counter 62004 – `BlocksReadCached` to counter 62002 – `BlocksRead` should be high, and counter 62006 – `SleepsWriteQ` should be relatively low. Such numbers would indicate that the SQM Writer is working approximately as fast as the SQM Reader, handing off blocks to one another without reading from disk. Remember, however, that this is a Replication Server-wide parameter—adjusting this parameter to make one queue more efficient may decrease the efficiency of another.

Caching system tables

Replication Server configuration parameters: `sts_cache_size` and `sts_full_cache_table_name`

You can fully cache certain system tables so that simple select statements on those tables do not require access to the RSSD. By default, `rs_repojbs` and `rs_users` are fully cached. Sybase recommends that you cache `rs_objects`, `rs_columns`, and `rs_functions`. Depending on the number of replication definitions and subscriptions used, fully caching these tables may significantly reduce RSSD access requirements. However, if the number of unique rows in `rs_objects` is approximately equal to the value for `sts_cachesize`, these tables may already be fully cached

Table 16-5 lists those tables that can be fully cached.

Table 16-5: System tables that can be cached

Tables

<code>rs_classes</code>	<code>rs_columns</code>	<code>rs_config</code>	<code>rs_datatype</code>
<code>rs_databases</code>	<code>rs_diskaffinity</code>	<code>rs_functions</code>	<code>rs_locator</code>
<code>rs_objects</code>	<code>rs_queues</code>	<code>rs_repdb</code>	<code>rs_routes</code>
<code>rs_sites</code>	<code>rs_systext</code>	<code>rs_translation</code>	<code>rs_users</code>
<code>rs_repojbs</code>	<code>rs_version</code>	<code>rs_publications</code>	<code>rs_dbreps</code>
<code>rs_dbsubsets</code>			

Setting wake up intervals

Replication Server configuration parameters: `rec_daemon_sleep_time`, `sub_daemon_sleep_time`, and `stats_daemon_sleep_time`

By default, the recovery and subscription daemons wake up every two minutes to check the RSSD for messages. In a typical production environment, the subscription daemon is used rarely. As a consequence, you may be able to set the subscription daemon wake-up interval to the maximum value: 31,536,000 seconds. Similarly, you can evaluate whether you want to set the recovery and statistics daemon to a longer wake-up interval.

Sizing the SQT cache

Replication Server configuration parameter: `sqt_max_cache_size`

Monitor SQT cache usage by reviewing counter 24005 – CacheMemUsed. Although this counter may indicate that the SQT cache is constantly full, `sqt_max_cache_size` may not need to be increased. Rather, monitor counter 24009 – TransRemoved. If TransRemoved remains zero, indicating that transactions are not being flushed from the cache to make room for others, `sqt_max_cache_size` need not be adjusted.

`sqt_max_cache_size` applies to all SQT caches supporting DIST clients, and provides a default value for SQT caches that support DSI clients. The DISTs can push through transactions rapidly; their SQT caches do not need to be as large as SQT caches for DSIs. Thus, it is advisable to set SQT cache sizes for DSIs individually using the connection configuration parameter `dsi_sqt_max_cache_size`, and using `sqt_max_cache_size` for DIST SQT caches only.

Note that counter 24010 – CacheExceeded does not provide useful information about the usage and size of the SQT cache.

Controlling the number of network operations

Database connection configuration parameter: `dsi_cmd_batch_size`

`dsi_cmd_batch_size` controls the size of a DSI command batch. That is, it controls the size of the buffer a DSI uses to send commands to a replicate data server. When the DSI configuration batch is set on, the DSI places as many commands as will fit into a single command batch before sending it to the replicate. In some cases, increasing the value of `dsi_cmd_batch_size` improves throughput by reducing the number of network operations.

You can monitor the average size of a batch by referring to counter 57078 – DSIEBatchSizeAve. You can monitor the average amount of time taken to process a batch (the time from when the batch is created until it is flushed and the results processed) by referring to counter 57074 – DSIEBatchTimeAve.

The following counters may also be useful in monitoring the effectiveness of batching and batch size:

57037 – SendTimeLast	57039 – SendTimeMax	57041 – SendTimeAvg
57063 – DSIEResultTimeLast	57065 – DSIEResultTimeMax	57067 – DSIEResultTimeAve
57069 – DSIEBatch	57070 – DSIEBatchTimeLast	57072 – DSIEBatchTimeMax
57076 – DSIEBatchSizeLast	57077 – DSIEBatchSizeMax	57079 – DSIEOCmdCountLast
57080 – DSIEOCmdCountMax	57081 – DSIEOCmdCountAve	57092 – DSIEBFMaxBytes

Controlling the number of outstanding bytes

Database connection configuration parameters: `exec_sqm_write_request_limit` and `md_sqm_write_request_limit`

`exec_sqm_write_request_limit` controls the maximum number of outstanding bytes the RepAgent User thread can hold before it must wait for some of those bytes to be written to the inbound queue. Similarly, `md_sqm_write_request_limit` controls the number of outstanding bytes a Distributor can hold before it must wait for some of those bytes to be written to the outbound queue.

Monitor the number of times and duration of time the RepAgent Executor sleeps while waiting for outstanding write requests to complete by reviewing these counters:

- 58018 – RWriteWaits
- 58019 – RAWWriteWaitsTimeAve

If the RepAgent Executor consistently reaches this threshold, review the StableDevice I/O.

Controlling the number of commands the RepAgent executor can process

Database connection configuration parameter: `exec_cmds_per_timeslice`

By default, the value of the `exec_cmds_per_timeslice` parameter is 5, which indicates that the RepAgent executor thread can process no more than five commands before it must yield the CPU to other threads. Depending on your environment, increasing or decreasing these values may improve performance.

If the in-bound queue is slow to be processed, try increasing these values to give the RepAgent executor thread and the distributor thread more time to perform their work. If, however, the out-bound queue is slow to be processed, try decreasing these parameter values so that the DSI has more time to work.

If CPU resources are limited with respect to the number of connections Replication Server supports, increasing the value of `exec_cmds_per_timeslice` may result in decreased overall performance. In this case, giving the RepAgent Executor more control of CPU resources may reduce resources to other Replication Server threads.

Monitor the number of times and duration of time the RepAgent Executor yields CPU with these counters:

- 58015 – RAYields
- 58016 – RAYieldTimeAve

Specifying the number of stable queue segments allocated

Replication Server configuration parameter: `sqm_recover_segs`

`sqm_recover_segs` specifies the number of stable queue segments Replication Server allocates before updating the RSSD with recovery QID information.

If `sqm_recover_segs` is set low, more RSSD updates are performed, possibly slowing performance. If `sqm_recover_segs` is set high, fewer RSSD updates are performed, possibly improving performance at the expense of longer recovery times.

Monitor how often an SQM Writer makes updates to the `rs_oqids` table by reviewing counter 6036 – `UpdsRsoqid`. Typically, increasing the value of `sqm_recover_segs` improves performance by reducing the amount of time and system resources necessary to allocate segments. However, queue startup and restart take longer as the SQM Writer must scan more of the queue to determine the last message successfully written for each origin. Each segment requires 1MB of queue space; determine the value of `sqm_recover_segs` by calculating the number of megabytes the SQM Writer can afford to scan at startup or restart. For example, if the SQM Writer can scan 50MB of queue without slowing Replication Server startup or restart, set `sqm_recover_segs` to 50.

Selecting disk partitions for stable queues

Database connection configuration parameter: `disk_affinity`

The Replication Server partition affinity feature (see “Allocating queue segments” on page 593) allows you to choose the disk partition to which Replication Server allocates segments for stable queues. Sybase suggests that to improve overall throughput you associate faster devices with stable queues that process more slowly.

Making SMP more effective

Replication Server configuration parameter: `smp_enable`

To determine the number of processors required to make effective use of SMP, establish a base of two processors plus one more for every four queues.

Processor speed may determine whether these numbers are correct to meet your performance needs. If you have outbound queues supporting parallel DSI, and there are more than 12 DSI Executor threads, you may want to increase the processor/thread ratio for outbound queues—one processor for every three or even two outbound queues.

Replication Server always uses a finite number of threads based on the number of supported connections and routes. Even if all threads are to be kept always busy, making more and more processors available to Replication Server will eventually cause “CPU saturation”—beyond which more processors will not increase performance. At that point, any performance issues you experience as a result of CPU resources may best be addressed by introducing CPUs running at faster speeds.

Although test results are inconclusive, there is evidence that making too many processors available to Replication Server can actually decrease performance. In such cases, the issue seems to be the amount of time taken to force thread context switches among the available processors. Use your operating system (OS) monitoring utilities to monitor the OS’s management of the Replication Server process and its threads. These utilities will help you determine if a reduction in CPUs made available to Replication Server reduces the number of such context switches.

Specifying the number of transactions in a group

You can use different configuration parameters to control the number of transactions in a group.

Database configuration parameter : `dsi_max_xacts_in_group`

`dsi_max_xacts_in_group` specifies the maximum number of transactions in a group. Larger numbers may reduce commit processing at the replicate database, and thereby improve throughput.

Monitor the average number of transactions placed in a group per DSI-E thread by reviewing counter 57049 – TransAvgGroup.

Monitor the average number of transactions per group for the total DSI connection by reviewing these counters:

- 5000 – TransTotal

- 5002 – NgTransTotal

Use `dsi_max_xacts_in_group` to control group size. Set `dsi_xact_group_size` to the maximum value and do not change it. Contention among parallel transactions may be reduced by reducing the value of `dsi_max_xacts_in_group` to 1, which indicates no grouping.

Monitor why groups are being closed by reviewing these counters:

- 5042 – GroupsClosedBytes
- 5043 – GroupsClosedNoneOrig
- 5044 – GroupsClosedMixedUser
- 5045 – GroupsClosedMixedMode
- 5049 – GroupsClosedTranPartRule
- 5051 – UserRuleMatchGroup
- 5053 – TimeRuleMatchGroup
- 5055 – NameRuleMatchGroup
- 5063 – GroupsClosedTrans
- 5068 – GroupsClosedLarge
- 5069 – GroupsClosedWSBSpec
- 5070 – GroupsClosedResume
- 5071 – GroupsClosedSpecial

Database configuration parameters: `dsi_xact_group_size` and `dsi_max_xacts_in_group`

Use these configuration parameters together to increase the number of transactions that can be grouped as a single transaction for application to the replicate database. If the average number of commands per transaction is small (five or fewer), you can use `dsi_xact_group_size` and `dsi_max_xact_in_group` to improve transaction application time.

Sybase recommends that you set `dsi_xact_group_size` to the maximum value, and use `dsi_max_xact_in_group` to control transaction group size.

Using parallel DSI threads

You can configure a database connection so that transactions are applied to a replicate data server using parallel DSI threads rather than a single DSI thread. Applying transactions in parallel increases the speed of replication, yet maintains the serial order of the transactions that occurred at the primary site.

When parallel DSI threads are active, Replication Server normally starts processing a transaction *before* the preceding transaction has committed and *after* the DSI has seen the commit record for the next transaction. The commit is delayed until it is determined that all preceding transactions have committed. Replication Server can control the order in which transactions are committed and detect conflicting updates in transactions that are executing simultaneously using either of these methods:

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

Replication Server can achieve additional parallelism in the way it processes large transactions with parallel DSI threads. Large transactions begin processing *before* the DSI has seen the commit record. While this means a large transaction can be processed sooner, it also means that Replication Server might start processing a transaction that is ultimately rolled back.

Replication Server provides other options for maximizing parallelism and minimizing contentions between transactions. For example:

- Transaction serialization methods allow you to choose the degree of parallelism your system can handle without inducing conflicts.
- Transaction partitioning rules provide additional tuning to affect how transactions are grouped and distributed.

Benefits and risks

For most primary databases, many users and applications can create transactions simultaneously. Funneling all of these transactions to the replicate through a single connection can create a serious bottleneck. This bottleneck can cause periods of unwanted latency between the primary and the replicate.

The benefit of enabling parallel DSI within Replication Server is to reduce this potential bottleneck by processing multiple transactions across multiple replicate connections at the same time.

The risk in enabling parallel DSI is the introduction of contention between the multiple replicate connections and their transactions. The simultaneous application of transactions against the replicate may introduce competition between the transactions for replicate resources, creating a different kind of bottleneck.

As a result, using parallel DSI threads successfully requires an in-depth knowledge of your replication environment and iterative testing to determine which of the parallel DSI tuning parameters are most beneficial. The objective is to provide high throughput while controlling the amount of contention introduced at the replicate.

For example, consider a body of work that includes 1000 transactions that must be replicated. It will take some time to send all 1000 transactions across a single replicate connection. However, attempting to configure and use 1000 connections, one for each transaction, will likely result in contentions and strained server resources. A successful configuration requires a balance between the two scenarios; it depends on both the transactions profile and the impact of issuing those transactions against the replicate using parallel DSI.

In a second example, two serial transactions issued at the primary each perform a single update operation to the same table row. If these two transactions are attempted in parallel at the replicate by two connections, the first transaction to access the table row is granted exclusive access. The second transaction must wait until the first transaction has either committed or rolled back and thus released the row. Although both transactions are ultimately applied, there is no benefit from the parallel DSI configuration. The transactions are processed serially in the same way they would have been processed without parallel DSI. The contention has nullified any benefit from using parallel DSI.

Parallel DSI parameters

You can customize the parallel DSI thread environment using the configuration parameters shown in Table 16-6. Use these configuration parameters with `alter connection` to tune parallel DSI threads for individual connections.

Table 16-6: Parallel DSI configuration parameters

Configuration parameter	Description
<code>dsi_commit_check_locks_intrvl</code>	<p>The number of milliseconds (ms) the DSI executor thread waits between executions of the <code>rs_dsi_check_thread_lock</code> function string.</p> <p>Default: 1000 ms (1 second)</p> <p>Minimum: 0</p> <p>Maximum: 86,400,000 ms (24 hours)</p>

Configuration parameter	Description
dsi_commit_check_locks_log	The number of times the DSI executor thread executes the <code>rs_dsi_check_thread_lock</code> function string before logging a warning message. Default: 200 Minimum: 1 Maximum: 1,000,000
dsi_commit_check_locks_max	The maximum number of times the DSI executor thread executes the <code>rs_dsi_check_thread_lock</code> function string before rolling back and retrying a transaction. Default: 400 Minimum: 1 Maximum: 1,000,000
dsi_commit_control	Specifies whether commit control processing is handled internally by Replication Server using internal tables (on) or externally using the <code>rs_threads</code> system table (off). Default: on
dsi_ignore_underscore_names	When the <code>dsi_partitioning_rule</code> is set to "name," specifies whether or not Replication Server ignores transaction names that begin with an underscore. Values are "on" and "off." Default: on
dsi_large_xact_size	The number of statements allowed in a transaction before it is considered to be a large transaction. Default: 100 Minimum: 4
dsi_num_large_xact_threads	The number of parallel DSI threads to be reserved for use with large transactions. The maximum value is one less than the value of <code>dsi_num_threads</code> . Default: 0
dsi_num_threads	The number of parallel DSI threads to be used for a connection. A value of 1 disables the parallel DSI feature. Default: 1 Minimum: 1 Maximum: 255
dsi_partitioning_rule	Specifies the partitioning rules (one or more) the DSI uses to partition transactions among available parallel DSI threads. Values are <code>origin</code> , <code>origin_sessid</code> , <code>time</code> , <code>user</code> , <code>name</code> , and <code>none</code> . See "Partitioning rules: reducing contention and increasing parallelism" on page 576 for detailed information. Default: none

Configuration parameter	Description
dsi_serialization_method	<p>Specifies the method used to maintain serial consistency between parallel DSI threads when applying transactions to a replicate data server.</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. isolation_level_3 – specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started, and that transaction isolation level 3 locking be used in the replicate data server. wait_for_commit – specifies that a transaction cannot start until the transaction scheduled to commit immediately preceding it is ready to commit. none – same as wait_for_start. Retained for backward compatibility. single_transaction_per_origin – same as wait_for_start with dsi_partitioning_rule set to origin. Retained for backward compatibility. <p>Default: wait_for_commit</p>
dsi_sqt_max_cache_size	<p>The maximum SQT cache size for the database connection. The default, 0, means the current setting of the sqt_max_cache_size parameter is used as the maximum cache size for the connection.</p> <p>Default: 0</p>
parallel_dsi	<p>A shorthand method for configuring parallel DSI to default values. A value of “on” sets dsi_num_threads to 5, dsi_num_large_xact_threads to 2, dsi_serialization_method to wait_for_commit, and dsi_sqt_max_cache_size to 1 million bytes. A value of “off” sets the parallel DSI values to their defaults. You can set this parameter to “on” and then set individual parallel DSI configuration parameters to fine-tune your configuration.</p> <p>Default: off</p>

To configure a connection for parallel DSI, set the parallel_dsi parameter to on and then set individual parallel DSI configuration parameters to fine-tune your environment.

For example, to enable parallel DSI for the connection to the pubs2 database on the SYDNEY_DS data server, enter:

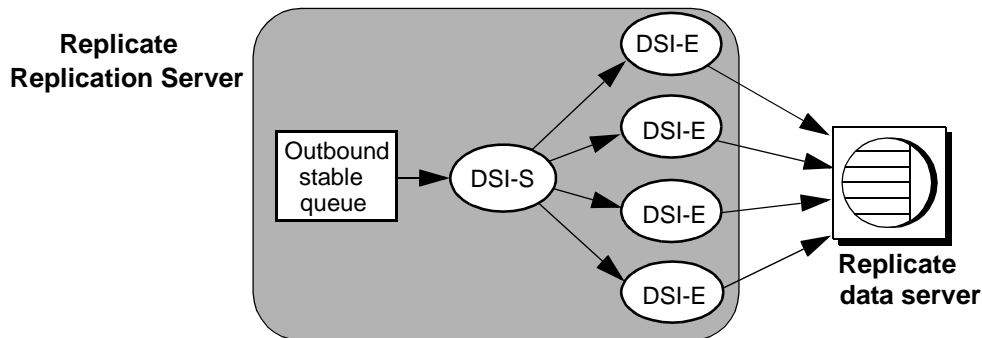
```
alter connection to SYDNEY_DS.pubs2
set parallel_dsi to 'on'
```

See “Configuring parallel DSI for optimal performance” on page 587 for guidelines on configuring the parameters.

Components of parallel DSI

Figure 16-3 shows the components of parallel DSI.

Figure 16-3: Parallel DSI components



DSI scheduler thread

The DSI scheduler thread (shown as DSI-S in Figure 16-3) collects small transactions into groups by commit order. Once transactions are grouped, the DSI scheduler dispatches the groups to the next available DSI executor thread. The DSI scheduler attempts to dispatch groups for different origins in parallel, because they can commit in parallel.

Transaction partitioning rules allow you to specify additional criteria the DSI scheduler can use to group transactions. See “Partitioning rules: reducing contention and increasing parallelism” on page 576.

DSI executor threads

The DSI executor threads (shown as DSI-E in Figure 16-3) map functions to function strings and execute the transactions on the replicate database. The DSI executor threads also take action on any errors the replicate data server returns.

Processing transactions with parallel DSI threads

You can define large and small transactions with the `dsi_large_xact_size` database connection configuration parameter. `dsi_large_xact_size` specifies the number of commands allowed in a transaction before the transaction is considered to be large. Replication Server normally processes small and large transactions differently.

Small transactions

Replication Server attempts to group similar transactions to process them as one, larger transaction. In this way, Replication Server can issue one commit for the group rather than committing each individual transaction. A group of transactions is complete and sent to the next available DSI executor thread when one of several criteria is met. For example:

- The next transaction has been issued from a different origin.
- The number of transactions in the group exceeds the value specified by `dsi_max_xacts_in_group`.
- The total size, in bytes, of the transactions in the group exceeds the value specified by `dsi_xact_group_size`.
- The next transaction is a large transaction, which is always grouped by itself.
- A transaction partitioning rule determines that the next transaction cannot be grouped with the existing group.
- A timeout expires.

Once a group is complete, it can be sent to the next available DSI executor thread. Only committed transactions can be added to a group. That is, transactions are not added to the small transaction group until their commit record is read.

Large transactions

Large transactions are submitted to the next available DSI executor thread that is reserved for a large transaction. The DSI executor thread sends the transaction to the replicate data server without waiting to see the commit record. If the transaction was rolled back at the primary data server, the DSI executor thread rolls it back at the replicate data server.

If Replication Server encounters a large transaction, and a dedicated large transaction thread is not available, the transaction is processed in the same way as a group of small transactions.

Transaction serialization methods

Replication Server maintains transaction serialization in parallel DSI by committing transactions at the replicate database in the same order in which they were committed at the primary database. When using more than one DSI thread to process transactions, Replication Server can execute operations in a different order at the replicate database as long as the order of processing does not result in conflicting updates. The commit order at the primary and replicate databases *always* matches; the execution or processing order at the replicate database may or may not match.

Replication Server provides four different serialization methods for minimizing update conflicts. The serialization method you choose depends on the amount of contention you expect between parallel threads and your replication environment. Each serialization method defines how much of a transaction can start before it must wait for the previous transaction to commit.

The serialization methods are:

- no_wait
- wait_for_start or none
- wait_for_commit
- single_transaction_per_origin

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

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

Note Non-Sybase data servers may not support the functionality of the `isolation_level_3` method. However, you can alter its associated function string, `rs_set_isolation_level3`, to accommodate the non-Sybase data server.

We can describe a transaction as containing three parts:

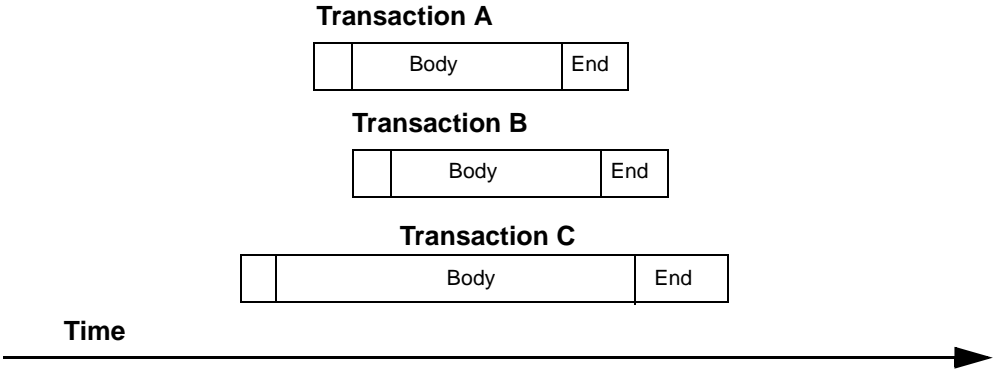
- The beginning
- The body of the transaction, consisting of operations such as insert, update, or delete
- The end of the transaction, consisting of a commit or a rollback

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

no_wait

This method instructs the DSI to initiate the next transaction without waiting for the previous transaction to commit. It assumes that your application is designed to avoid conflicting updates, or that `dsi_partitioning_rule` is used effectively to reduce or eliminate contention. Adaptive Server does not hold update locks unless transaction isolation level 3 has been set. The method assumes little contention between parallel transactions and results in the nearly parallel execution shown in Figure 16-5.

Figure 16-4: Thread timing with the no_wait serialization method



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

wait_for_start

Note For clarity, the `wait_for_start` option replaces the `none` option in Replication Server 12.6. `none` is retained for backward compatibility.

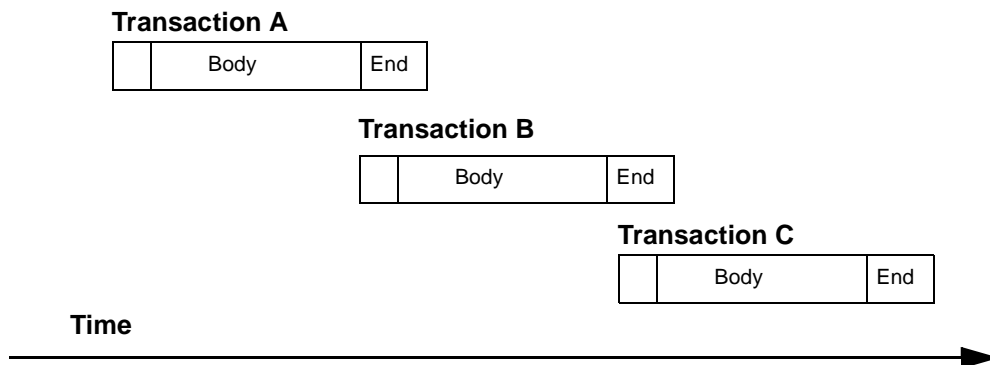
`wait_for_start` specifies that a transaction can start as soon as the transaction scheduled to commit immediately before it has started.

When `wait_for_start` is used along with the origin partitioning parameter, transactions from each origin are sent to separate DSI executor threads. This method assumes that multiple origins updating the same replicate have built-in contention control, so they can be safely applied in parallel to the replicate.

`wait_for_commit`

In this method, the next thread's transaction group is not sent for processing until the previous transaction has processed successfully and the commit is being sent. This is the default setting. It assumes considerable contention between parallel transactions and results in the staggered execution shown in Figure 16-5.

Figure 16-5: Thread timing with `wait_for_commit` serialization method



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

`wait_for_commit`, because it provides minimal parallelism, cannot deliver the most improved performance. However, it also has less likelihood of creating contentions.

`isolation_level_3`

This method is identical to `wait_for_start` except that it specifies transaction isolation level 3 for the DSI /E session in the replicate database. This type of locking prevents “non-repeatable reads” and “phantoms” from occurring by applying an index page or table lock until the end of the transaction.

Select this serialization method if you are using triggers to enforce referential integrity of data across a database. It prevents phantom rows from occurring in a table while a trigger is being executed.

Replication Server uses the `rs_set_isolation_level_3` system function to turn on transaction `isolation_level_3`. This system function is executed every time the DSI connects to the replicate data server.

Use this method only when `wait_for_start` does not introduce inordinate contention and the replicated transaction must perform a repeatable read, or the replicate database is using row-level locking.

Note Replication Server automatically creates a function string for the `rs_set_isolation_level3` function in function-string classes in which Replication Server generates default function strings. For other function-string classes, you must create `rs_set_isolation_level3` function strings before you can use the parallel DSI feature.

single_transaction_per_origin

This method attempts to send the transactions from each origin to separate DSI executor threads. It assumes that multiple origins updating the same replicate have built-in contention control, so they can be safely applied in parallel to the replicate.

Note With Replication Server 12.6, the functionality of `single_transaction_per_origin` is duplicated by selecting the `wait_for_start` serialization method and the origin partitioning rule. `single_transaction_per_origin` has been retained for backward compatibility. See “`wait_for_start`” on page 573.

Partitioning rules: reducing contention and increasing parallelism

Another parallel DSI tuning parameter is `dsi_partitioning_rule`. Partitioning rules set using `dsi_partitioning_rule` allow the parallel DSI feature to make decisions about transaction groups and parallel execution based on transactions having common names, users, overlapping begin/commit times, or a combination of these. Partitioning rules allow the parallel DSI feature to more closely mimic processing order at the primary.

Each of the parallel DSI parameters provides a method for fine-tuning the feature based on conditions at your installation. `dsi_num_threads` controls the number of DSI threads available for a connection. `dsi_serialization_method` controls the amount of parallelism for the connection, but must balance increased parallelism with the potential for contentions at the replicate. `dsi_partitioning_rule` provides a method for reducing contentions without reducing the overall capabilities of the parallel DSI feature.

Using transaction-partitioning rules

Replication Server allows you to partition transactions for each connection according to one or more of these attributes:

- Origin
- Origin and session ID
- None, in which no partitioning rule is applied
- User name
- Origin begin and commit times
- Transaction name

Note If partitioning rules are to be used to improve performance, `dsi_serialization_method` must not be `wait_for_commit`. `wait_for_commit` removes contention by reducing parallelism.

To select partition rules, use the `alter connection` command with the `dsi_partitioning_rule` option. The syntax is:

```
alter connection to data_server.database
  set dsi_partitioning_rule to '{ none|rule[, rule ]}'
```

Values for *rule* are `user`, `time`, `origin`, `origin_sessid`, and `name`.

For example, to partition transactions according to user name and origin begin and commit times, enter:

```
alter connection to TOKYO_DS.pubs2
    set dsi_partitioning_rule to 'user,time'
```

Partitioning rule: origin

origin causes transactions from the same origin to be serialized when applied to the replicate database .

Partitioning rule: origin and process ID

origin_sessid causes transactions with the same origin *and* the same process ID to be serialized when applied to the replicate database. Sybase recommends that when first trying partitioning rules start with a setting of `origin_sessid,time`.

Note The process ID for Application Server is the Session Process ID (SPID).

Partitioning rule: none

none is the default behavior, in which the DSI scheduler assigns each transaction group or large transaction to the next available parallel DSI thread.

Partitioning rule: user name

If you choose to partition transactions according to user name, transactions entered by the same primary database user ID are processed serially. Only transactions entered by different user IDs are processed in parallel.

Use of this partitioning rule avoids contentions, but may in some cases cause unnecessary loss of parallelism. For example, consider a DBA who is running multiple batch jobs. If the DBA submits each batch job using the same user ID, Replication Server processes each one serially.

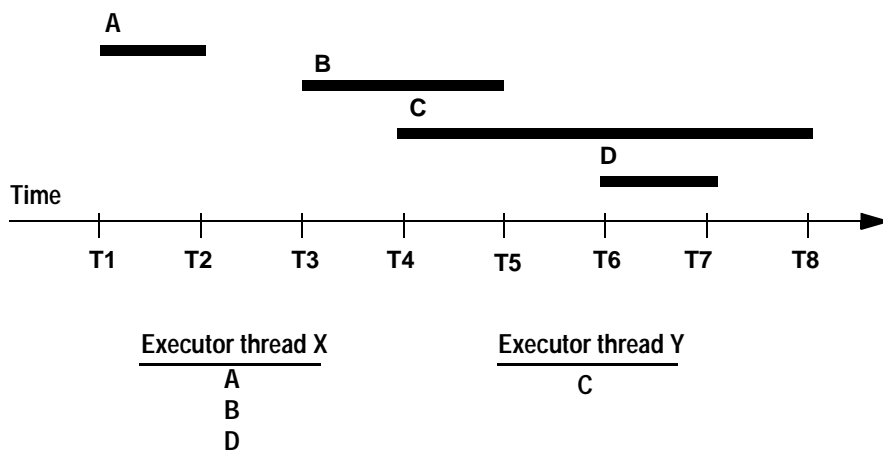
The user name partitioning rule is most useful if each user connection at the primary has a unique ID. It is less useful if multiple users log on using the same ID, such as “sa.”

Partitioning rule: origin begin and commit times

If the time partitioning rule is used, the DSI scheduler looks at the origin begin and commit times of transactions to determine which transactions could *not* have been executed by the same process at the primary database. A transaction whose origin begin time is earlier than the commit time of the preceding transaction can be processed by a different DSI executor thread.

Suppose the origin begin and commit times partitioning rule has been selected, and the transactions and processing times shown in Figure 16-6 are all from the same primary database.

Figure 16-6: Transaction origin begin and commit times



In this example, the DSI scheduler gives transaction A to DSI executor thread X. The scheduler then compares the begin time of transaction B and the commit time of transaction A. As transaction A has committed before transaction B begins, the scheduler gives transaction B to executor thread X. That is, transactions A and B may be grouped together and may be processed by the same DSI executor thread. Transaction C, however, begins before transaction B commits. Therefore, the scheduler assumes that transactions B and C were applied by different processes at the primary, and gives transaction C to executor thread Y. Transactions B and C are not allowed in the same group and may be processed by different DSI executor threads. Because transaction D begins before transaction C commits, the scheduler can safely give transaction D to executor thread X.

Note Use of the origin begin and commit times partitioning rule may lead to contentions when large transactions are processed, as they are scheduled before the commits are seen.

Partitioning rule: transaction name

The DSI scheduler can use transactions names to group transactions for serial processing. When creating a transaction on Adaptive Server, you can use the begin transaction command to assign a transaction name.

If the transaction name partitioning rule is applied, the DSI scheduler assigns transactions with the same name to the same executor thread. Transactions with different transaction names are processed in parallel. Transactions with a null or blank name are ignored by the name parameter. Their processing is determined by other DSI parallel processing parameters or the availability of other executor threads.

Note This partitioning rule is available to non-Sybase data servers only if they support transaction names.

Default transaction names

By default, Adaptive Server always assigns a name to each transaction. If a name has not been assigned explicitly using begin transaction, Adaptive Server assigns a name that begins with the underscore character and includes additional characters that describe the transaction. For example, Adaptive Server assigns a single insert command the default name “_ins.”

Use the `dsi_ignore_underscore_name` option with `alter connection` to specify whether or not Replication Server ignores these names when partitioning transactions based on transaction name. By default, `dsi_ignore_underscore_name` is on, and Replication Server treats transactions with names that begin with an underscore in the same way it treats transactions with null names.

Using multiple transaction rules

You can set multiple transaction rules for a single connection. For example, applying both `origin session ID` and `origin begin and commit times` best approximates the processing environment at the primary database.

When more than one transaction rule is specified, Replication Server applies the rules in the order in which they are entered in the `alter connection set dsi_partitioning_rule` syntax.

For example, if `dsi_partitioning_rule` is set to “time, user,” Replication Server checks `origin begin and commit times` before checking user ID. If no conflict exists for `origin begin and commit times`, Replication Server checks user ID. If there is a conflict involving `begin and commit times`, Replication Server applies the time rule without checking the user ID. Thus, two transactions will be assigned to different parallel DSI threads if the `origin begin time` of the later transaction is earlier than the `commit time`—even if both transactions have the same user ID.

Grouping logic and transaction partitioning rules

Partitioning rules can affect grouping as well as scheduling decisions. When no partitioning rule is applied, a group is complete when, for example, the maximum size for a group is reached or a large transaction is encountered.

If a partitioning rule determines that two transactions occurred at overlapping times (time rule), have different transaction names (name rule), or are from different users (user rule), the two transactions are not allowed in the same group. Otherwise, normal group-size decisions are applied, based on transaction size, origin, and so forth. See “Small transactions” on page 571.

Resolving conflicting updates

Parallel DSI processing must duplicate the commit order of transactions at the primary database yet allow transaction updates to process simultaneously. It must then resolve any transaction contentions that occur as a result. Critical transaction contentions—or contention deadlocks—can occur when a transaction cannot commit because it must wait for an earlier transaction to commit, and the earlier transaction cannot commit because needed resources are locked by the later transaction.

For example, DSI threads A and B are processing transactions in parallel. Thread A's transaction must commit before thread B's transaction. Thread B's transaction locks resources needed by thread A. Thread B's transaction cannot commit until thread A's transaction commits, and thread A's transaction cannot commit because needed resources are locked by thread B.

Replication Server provides two methods for resolving deadlocks:

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

The internal method is handled primarily within Replication Server, and uses the `rs_dsi_check_thread_lock` function string for deadlock detection. The external method requires both Replication Server and the replicate database, and uses the `rs_threads` system table for both commit order validation and deadlock detection.

Sybase recommends the internal method, which is the default, for both Sybase and non-Sybase data servers. This method requires less network I/O than the external method, and, if a deadlock occurs, may require the rollback of only a single transaction. The external method requires more network I/O and results in the rollback of several transactions. The external method is included for compatibility with previous versions of Replication Server.

If Replication Server encounters commit contention and `dsi_commit_control` is "on," Replication Server rolls back and retries one transaction. If, however, Replication Server encounters transaction contention and `dsi_commit_control` is "on," Replication rolls back and retries all transactions serially. If `dsi_commit_control` is "off," Replication Server rolls back and retries all transactions serially for either type of contention.

To select a method, enter the `alter connection` command with the `dsi_commit_control` option. For example, to choose the internal method for the `pubs2` database on the `TOKYO_DS` data server, enter:

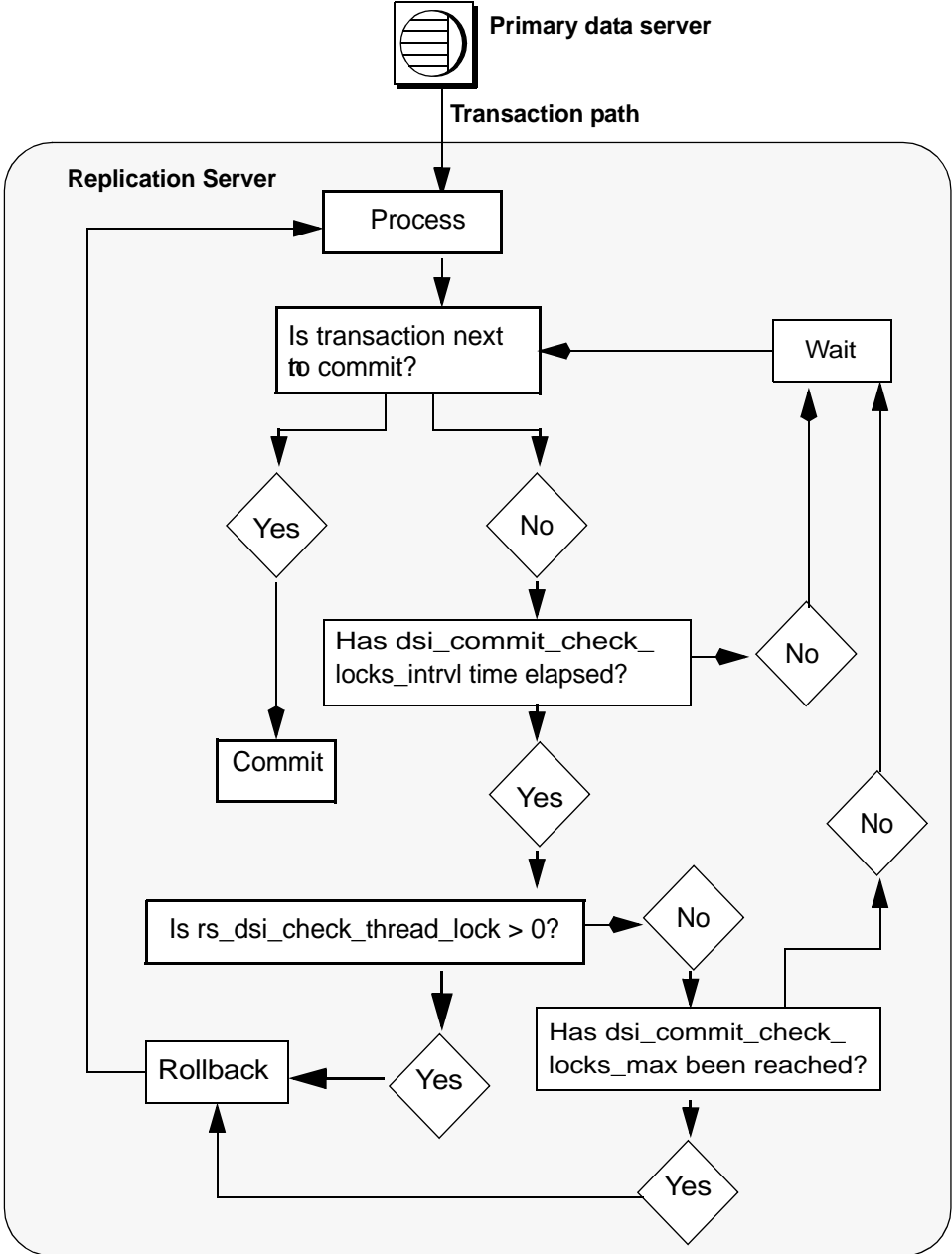
```
alter connection to TOKYO_DS.pubs2
set dsi_commit_control to 'on'
```

Setting `dsi_commit_control` to “on” specifies the internal method; setting `dsi_commit_control` to “off” specifies the external method.

Resolving conflicts internally using the `rs_dsi_check_thread_lock` function string

To resolve conflicting updates, Replication Server must maintain transaction commit order and resolve commit consistency deadlocks. Figure 16-7 describes the logic Replication Server uses to resolve conflicts using the `rs_dsi_check_thread_lock` function string.

Figure 16-7: Conflict resolution logic using the `rs_dsi_check_thread_lock` function string



Note The internal method assumes that Replication Server detects and resolves conflicting updates within Replication Server. If a deadlock is detected by the replicate database, the replicate chooses a transaction to roll back. To guarantee commit order, Replication Server must roll back all transactions currently executing against the replicate database. Replication Server then reapplies the transactions serially.

Maintaining commit order

Replication Server reads the commit information sent from the primary database and uses this information to define and maintain the transaction commit order at the replicate database.

If a DSI executor thread's transaction processing is complete and it is expected to be the "next" transaction to commit, it is allowed to do so. If a thread's transaction processing is complete and it is *not* the "next" transaction expected to commit, the thread must await its turn to commit.

Resolving commit consistency deadlocks

If a thread's transaction processing is complete and it is not the next transaction expected to commit, it could be holding resources required by an earlier transaction. After waiting the amount of time specified in the `dsi_commit_check_locks_intrvl` parameter, a DSI executor thread executes the `rs_dsi_commit_check_thread_lock` function string to determine if the thread holds a lock on resources needed by an earlier transaction:

- If the thread is blocking another transaction (`rs_dsi_check_thread_lock > 0`), the current transaction rolls back, which resolves the deadlock and allows the earlier transaction to commit. Only the blocking transaction rolls back; other transactions process normally.
- If the thread is not blocking another transaction, it checks to see if it has executed `rs_dsi_check_thread_lock` more times than is defined by the `dsi_commit_check_locks_max` parameter.
 - If the thread has not executed `rs_dsi_check_thread_lock` more times than is defined in `dsi_commit_check_locks_max`, the transaction commits if it is next, or it waits again the amount of time specified in `dsi_commit_check_locks_intrvl`.
 - If the thread has executed `rs_dsi_check_thread_lock` more times than is defined in `dsi_commit_check_locks_max`, the current transaction rolls back.

Function strings for internal commit control

Replication Server uses the `rs_dsi_check_thread_lock` function to check whether the current DSI executor thread is blocking another replicate database process. This function has function-string-class scope. It is called only if the DSI executor thread is ready to commit but cannot because it is not next to commit, and the amount of time specified for `dsi_commit_check_locks_intrvl` has elapsed.

Table 16-7: System functions that support internal commit control

Function	Description
<code>rs_dsi_check_thread_lock</code>	Determines whether or not the DSI executor thread is holding a lock that blocks a replicate database process. A return value greater than 0 indicates that the thread is holding resources required by another database process, and that the thread should roll back and retry the transaction.

Note Replication Server automatically creates function strings for the above function in function-string classes in which Replication Server generates default function strings. For other function-string classes, you must create these function strings before you can use parallel DSI features with `dsi_commit_control` set on.

Using `rs_threads` to resolve conflicts externally

The `rs_threads` table is located in the replicate database. It contains a row for each DSI executor thread. To simulate row-level locking, it has two columns, `id` and `seq`, and enough dummy columns so that only one row fits on a page. The `id` column is used as a unique clustered index.

At the beginning of a transaction, the DSI executor thread updates its row in the `rs_threads` table with the next available sequence number. When it is ready to commit the transaction, the thread sends a `select` statement to the replicate data server to select, from the `rs_threads` table, the sequence number of the transaction that should have committed prior to the transaction.

Because the preceding transaction holds a lock on this row in `rs_threads`, this thread is blocked until the preceding transaction commits.

If the sequence number that is returned is less than the expected value, the thread determines whether it should roll back the transaction or retry the select operation. Because the DSI formats many commands into a single batch before submitting it to the Adaptive Server, a thread may be ready to commit before the preceding transaction has submitted any commands to the Adaptive Server. In this case, the select in the `rs_threads` table may be submitted several times.

If the sequence number that is returned matches the expected value, the transaction can commit.

Handling deadlocks

If a transaction is ready to commit, but cannot because it is not next in proper commit order, and this transaction is holding locks on resources that are needed by a transaction that must commit before this one, a deadly embrace occurs at the replicate database. The deadly embrace consists of the lock on `rs_threads` held by the next transaction in commit order, and the locks held on resources needed by that transaction. The deadly embrace is detected by the replicate database, which chooses a transaction to roll back. Since Replication Server must guarantee commit order, when this rollback is forced by the replicate database, Replication Server rolls back all transactions executing against the replicate database and reapplies them serially in commit order.

Function strings for commit control using `rs_threads`

Replication Server manipulates the `rs_threads` system table with the system functions listed below. These functions have function-string-class scope. They are executed only when more than one DSI thread is defined for a connection.

Table 16-8: System functions that modify the `rs_threads` system table

Function	Description
<code>rs_initialize_threads</code>	Sets the sequence of each entry in the <code>rs_threads</code> system table to 0. This function is executed during the initialization of a connection.
<code>rs_update_threads</code>	Updates the sequence number for the specified entry in the <code>rs_threads</code> system table.
<code>rs_get_thread_seq</code>	Returns the current sequence number for the specified entry in the <code>rs_threads</code> system table.
<code>rs_get_thread_seq_noholdlock</code>	Returns the current sequence number for the specified entry in the <code>rs_threads</code> system table, using the <code>noholdlock</code> option. This thread is used when the serialization method is isolation level 3.

Note The function strings described in Table 16-8 are needed only when the external, `rs_threads` method is used for commit control.

Configuring parallel DSI for optimal performance

The following guidelines can help you configure parallel DSI to achieve optimal performance. The objective is to tune parallel DSI processing to provide the best replication performance, balancing parallel processing with acceptable levels of contention. Contentions will always occur. The only way to eliminate contentions is to turn off parallel DSI processing. At the same time, setting all parallel DSI parameters for maximum parallelism may cause Replication Server to spend more time recovering from contentions than actually applying transactions to the replicate. Optimal performance is achieved through a clear understanding of your operating environment so that you can successfully balance parallel processing with acceptable contention levels.

Before you begin

Before you begin tuning for performance:

- *Understand your transaction profile.* What kinds of transactions are being replicated? Do these transactions affect the same rows and tables? Are these transactions liable to conflict if applied in parallel? Is the transaction profile constant, or does it change, perhaps with the time of day or month. A clear understanding of your transaction profile helps you select those parameters and settings that will be most useful.
- *Tune the replicate database to handle contentions.* Most primary databases have been tuned to minimize contentions through the use of clustered indexes, partitioning, row-level locking, and so on. Make sure that your replicate database has been tuned similarly.
- *Define a set of repeatable transactions that accurately reflect your replication environment.* Tuning your parallel DSI environment is an iterative process. You will need to set parameters, run a test, measure performance, compare against previous measurements, and repeat until you have maximized your results.

- *First, reset the `dsi_serialization_method` parameter.* Set the `dsi_serialization_method` parameter to `no_wait` to enable maximum parallelism. Then attempt to reduce contentions by testing other parameters. Because the `wait_for_commit` (the default) setting supplies minimal parallelism and therefore minimal benefit, only reset `dsi_serialization_method` to `wait_for_commit` after all attempts to reduce contention using the `no_wait` setting have failed to increase performance.
- *Set the `dsi_num_threads` parameter correctly.* The `dsi_num_threads` parameter defines the total number of DSI executor threads; the `dsi_num_large_xact_threads` parameter defines the total number of DSI executor threads reserved for large transactions. Thus, the total number of DSI executor threads (`dsi_num_threads`) equals the number of DSI threads reserved for large transactions plus the number of threads available for small transactions. Make sure that `dsi_num_threads` is greater than the default, and that the value for `dsi_num_threads` is greater than that for `dsi_num_large_xact_threads`.

Reducing contention

Start tuning parallel DSI parameters to reduce contention when you have completed the tasks described in “Before you begin” on page 587 and performance tests indicate that contentions are affecting performance. For example:

- The replicate is locking activity.
- Replication Server is rolling back and reapplying a large percentage of transactions due to deadlock conditions.

Start by tuning the `dsi_max_xacts_in_group` parameter, which determines the number of transactions grouped in a single begin/commit block. By reducing the value of `dsi_max_xacts_in_group`, you cause the DSI executor threads to commit more frequently. Thus, the DSI executor threads hold fewer replicate resources for shorter periods of time and contentions should decrease.

Adjusting the `dsi_num_threads` parameter also affects contention. The larger the number of DSI executor threads available, the more likely contentions will arise among the threads. Try decreasing the value of `dsi_num_threads` even to 3 with one reserved for large transactions. Finding the values that provide best performance is iterative. Remember that some contention is acceptable if overall performance improves.

Using partitioning rules

Partitioning rules can also reduce contention, but require a clear understanding of your transaction profile.

The transaction name rule

Do transactions have transaction names? Is the contention caused by transactions with the same name? Try setting the transaction name rule, which forces transactions with the same name to be sent to the replicate one-by-one.

If transactions are not named, you could change the application so that names are added. Then use the name rule to serialize only specified transactions. Suppose a particular type of large transaction always causes problems if the DSI executor threads attempt to process two or more in parallel. By giving the problem transactions the same name, and applying the name rule, you can ensure that the problem transactions are processed serially. Remember, however, that the name rule is applied to all transactions, and all transactions with the same name will be processed serially.

The user name rule

Setting the user name rule may help reduce contentions caused by transactions processed in parallel from the same user ID. Like the transaction name rule, the user name rule, if set, is applied to all transactions, and every transaction from the same user ID will be processed serially.

The origin begin and commit times rule

The time rule forces serial execution of transactions with nonoverlapping commit/begin times. That is, if the commit time of the first transaction comes before the begin time of the next transaction, these two transactions must execute serially.

Combining partition rules

You can combine rules. The first rule to be satisfied takes precedence. Thus, if, for example, the `origin_sessid`, time rule is specified, two transactions with the same origin session ID will be forced to run serially, and the time rule is not applied.

Frequent conflicting updates

If your transactions conflict with each other frequently, set the parallel DSI configuration parameters as follows:

- `dsi_serialization_method` – set this parameter to `wait_for_commit`.
- `dsi_num_large_xact_threads` – set this parameter to 2. If you are configuring parallel DSI in a warm standby application, set the `dsi_num_larg_xact_threads` parameter for the standby database to one more than the number of simultaneous large transactions executed at the active database.
- `dsi_num_threads` – set this parameter to 3 plus the value of the `dsi_num_large_xact_threads` parameter. If your transactions are usually small, such as one or two statements, set `dsi_num_threads` to 1 plus the value of `dsi_num_large_xact_threads`.

Setting the `parallel_dsi` configuration parameter on provides a shorthand method for configuring parallel DSI as described above. It also sets the `dsi_sqt_max_cache_size` parameter to 1 million bytes.

Infrequent conflicting updates

If your transactions conflict with each other only occasionally, set the parallel DSI configuration parameters as follows:

- `dsi_serialization_method` – set this parameter to isolation level 3 if you have Adaptive Server or SQL Server version 10.0.x or later.
- `dsi_num_large_xact_threads` – set this parameter to 2. If you are configuring parallel DSI in a warm standby application, set the `dsi_num_larg_xact_threads` parameter for the standby database to one more than the number of simultaneous large transactions executed at the active database.
- `dsi_num_threads` – set this parameter to 3 plus the value of the `dsi_num_large_xact_threads` parameter.

Transaction size and SQT cache

The `sqt_max_cache_size` configuration parameter determines the maximum SQT cache memory, in bytes. You set its value based on the number of transactions you are processing and the number of statements within the transactions.

The following formula gives the upper bound, or worst case, estimate for SQT cache size. A good default value for the SQT cache when you are using parallel DSI is 2MB.

The SQT cache formula is:

$$sqt_max_cache_size = T * (O + (S * N))$$

The factors included in the formula are as follows:

- *T* is the number of transactions to cache for a DSI. It should be 1 plus the number of ordinary DSI threads, multiplied by 20:

$$20 * (1 + dsi_num_threads - dsi_num_large_xact_threads)$$
- *O* is the SQT cache's per transaction overhead, including the begin and commit statements. This is 3K bytes.
- *S* is the statement size within the SQT cache. This is 1K when the modified data is less than 100 bytes, 2K when the modified data is between 100 and 300 bytes, and 5K when the modified data is greater than 300 bytes.
- *N* is the number of statements modified by a transaction. It is determined by the application.

When a steady stream of transactions is flowing to the DSI, the SQT should be full and the number of closed transactions should be about 20. More closed transactions indicates that the cache is larger than required, and fewer indicates that the cache is not large enough.

Number of statements and SQT cache

The *dsi_large_xact_size* parameter setting should be larger than the number of statements modified by an ordinary transaction and larger than the number given by the formula below. If the number below is smaller than the number of statements in an ordinary transaction, the *sqt_max_cache_size* parameter should be increased. The formula is:

$$dsi_large_xact_size = ((sqt_max_cache_size / dsi_num_large_xact_threads) - O) / S$$

Parallel DSI and the *rs_origin_commit_time* system variable

The value of the *rs_origin_commit_time* system variable depends on whether you are using the parallel DSI feature.

- If you are not using parallel DSI to process large transactions, the value of *rs_origin_commit_time* contains the time when the last transaction in the transaction group committed at the primary site.
- If you are using parallel DSI to process large transactions (before their commit has been read from the DSI queue), when the DSI threads start processing one of these transactions, the value of *rs_origin_commit_time* is set to the value of *rs_origin_begin_time*.

When the commit statement for the transaction is read, the value of *rs_origin_commit_time* is set to the actual commit time. Therefore, when the configuration parameter *dsi_num_large_xact_threads* is set to a value greater than zero, the value for *rs_origin_commit_time* is not reliable for any system function other than *rs_commit*.

Using multiprocessor platforms

You can run Replication Server on symmetric multiprocessor (SMP) or single-processor platforms. Replication Server's multithreaded architecture supports both hardware configurations. On a single processor platform, Replication Server's threads run serially. On a multiprocessor platform, Replication Server's threads can run in parallel, thereby improving performance and efficiency.

Replication Server is an Open Server application. Replication Server support for multiple processors is based on Open Server support for multiple processors. Both servers use the POSIX thread library on UNIX platforms and the WIN32 thread library on Windows platforms. For detailed information about Open Server support for multiple processing machines, see the *Open Server Server-Library/C Reference Manual*.

When Replication Server is in single-processor mode, a server-wide mutual exclusion lock (mutex) enforces serial thread execution. Serial thread execution safeguards global data, server code, and system routines, ensuring that they remain thread-safe.

When Replication Server is in multiprocessor mode, the server-wide mutex is disengaged and individual threads use a combination of thread management techniques to ensure that global data, server code, and system routines remain secure.

Enabling multiprocessor support

To specify whether Replication Server takes advantage of a multiprocessor machine, use `with` with the `smp_enable` option. For example:

```
configure replication server set smp_enable to 'on'
```

Setting `smp_enable` to “on” specifies multiprocessor support; setting `smp_enable` to “off” specifies single-processor support. The default is “off.”

`smp_enable` is a static option. You must restart Replication Server after changing the status of `smp_enable`.

Monitoring thread status

You can verify Replication Server thread status using these commands:

- `admin who` – provides information on all Replication Server threads
- `admin who_is_up` or `admin who_is_down` – lists Replication Server threads that are running, or not running.
- `sp_help_rep_agent` – provides information on the RepAgent thread and the RepAgent User thread.

See Chapter 13, “Verifying and Monitoring Replication Server” for more information about monitoring thread status.

Monitoring performance

Replication Server provides monitors and counters for monitoring performance. See Chapter 15, “Using Counters to Monitor Performance,” in the *Administration Guide*.

Allocating queue segments

You can choose the disk partition to which Replication Server allocates segments for stable queues. By choosing the stable queue placement, you can enhance load balancing and read/write distribution.

Replication Server stores messages destined for other sites on partitions. It allocates space in partitions to stable queues and operates in 1MB chunks called segments. Each stable queue holds messages to be delivered to another Replication Server or to a data server. The queues hold data until it is sent to its destination.

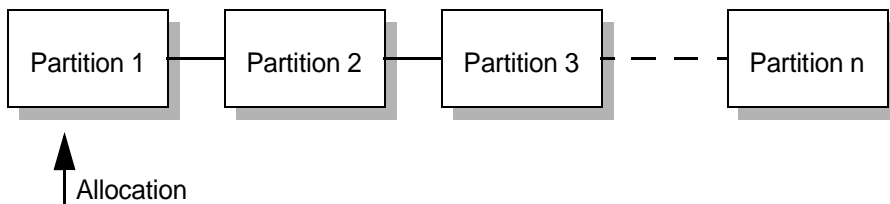
rs_init assigns Replication Server's initial partition. You may need additional partitions, depending on the number of databases and remote Replication Servers to which the Replication Server distributes messages.

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

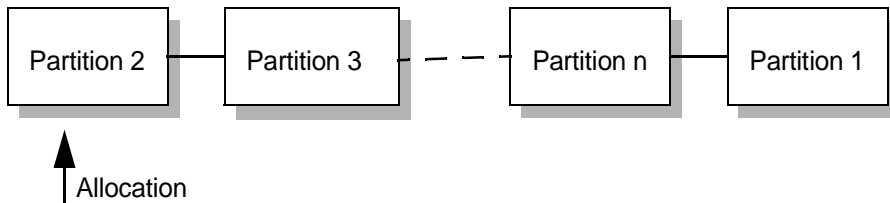
By default, Replication Server assigns queue segments to the first partition in an ordered list of partitions. See Figure 16-8. When the first partition becomes full, the first partition becomes the last partition, and the next queue segment is allocated to the new first partition. When the default method is used, the rolling allocation of segments is automatic and cannot be controlled by the user.

Figure 16-8: Default allocation mechanism

First allocation



Second allocation



Choosing disk allocations

To choose the segment allocation, use the alter connection or alter route command with the “set disk_affinity” option. The syntax is:

```
alter connection to dataserver.database
  set disk_affinity to [ 'partition' | 'off' ]

alter route to replication_server
  set disk_affinity to [ 'partition' | 'off' ]
```

partition is the logical name of the partition to which you want to allocate the next segment for the connection or route.

Each allocation directive is called a “hint” because Replication Server can override the allocation if, for example, the allocated partition is full or has been dropped. If Replication Server overrides the hint, it allocates segments according to the default mechanism described in Figure 16-8.

Replication Server checks for an allocation hint each time it allocates a new segment for a queue. Each hint is stored in the rs_diskaffinity system table. Each partition may have many hints, but each stable queue can have only one hint.

Successfully using disk allocation to improve performance depends on the architecture and other characteristics of your site. Sybase suggests that one way to improve overall throughput is to associate faster devices with those stable queues that process more slowly.

An example

You can allocate different disk partitions to different stable queues. You could, for example, make partitions of different sizes available to different database connections. In this example, we add partitions of 10MB and 20MB to the Replication Server and specify allocation hints for the TOKYO_DS and SEATTLE_DS data servers. The procedure is:

- 1 Make the partitions P1 and P2 on the device named /dev/rds0a available to Replication Server, enter:

```
add partition P1 on '/dev/rds0a' with size 20
add partition P2 on '/dev/rds0a' with size 10
```

- 2 Suspend the connection to the TOKYO_DS and SEATTLE_DS data servers, enter:

```
suspend connection to TOKYO_DS
suspend connection to SEATTLE_DS
```

- 3 Specify allocation hints for the connection to the TOKYO_DS and SEATTLE_DS data servers, enter:

```
alter connection to TOKYO_DS.db1
set disk_affinity to 'P1'

alter connection to SEATTLE_DS.db5
set disk_affinity to 'P2'
```

- 4 Resume the connections to the TOKYO_DS and SEATTLE_DS data servers, enter:

```
resume connection to TOKYO_DS
resume connection to SEATTLE_DS
```

Dropping hints and partitions

You can remove an allocation hint using the alter connection or alter route command with the set disk_affinity to 'off' parameter. For example:

```
alter connection to TOKYO_DS.db1
set disk_affinity to 'P1' to 'off'
```

This command deletes the allocation hint for P1 from the rs_diskaffinity table.

You can remove a partition from Replication Server using the drop partition command. If the partition you are dropping has one or more allocation hints in the rs_diskaffinity table, Replication Server marks the allocation hints for deletion, but does not delete them until all data stored on the partition has been successfully delivered and the partition has been dropped.

Using Counters to Monitor Performance

This chapter describes Replication Server counters that monitor performance. “Using counters to monitor RepAgent performance” on page 134 describes RepAgent counters that monitor performance.

Topic	Page
Introduction	597
Modules and counters: an overview	598
Enabling sampling and flushing	600
Configuring modules, connections, and routes	603
Viewing information about the counters	605
Viewing current counter values	606
Viewing values flushed to the RSSD	607
Resetting counters	608
dSTATS daemon thread	608

Introduction

Replication Server gathers statistics using over a hundred different counters. By default, most counters are active, and current values can be viewed at any time using the `admin statistics` command. Counter values can also be flushed (or saved) to the RSSD, where they can be calculated and viewed using standard Transact-SQL statements.

Replication Server does not automatically flush statistics to the RSSD because it would fill up the RSSD quickly. The counter configuration is designed to limit the output sent to the RSSD. Several commands and configuration parameters allow you to control which counter metrics are flushed to the RSSD and when.

The thread daemon, `dSTATS`, controls flushing. Three system tables contain information about counters:

- `rs_statcounters` – contains descriptive information about each counter. Values in this table do not change.
- `rs_statrun` – describes each sampling period (or run).
- `rs_statdetail` – contains metrics from each run.

See the *Replication Server Reference Manual* for information about system tables.

Modules and counters: an overview

Replication Server runs multiple threads concurrently. The total number of threads depends on the number of databases that a Replication Server manages and the number of Replication Servers to which it has direct routes. Each thread performs a specific function such as managing a user session or applying transactions to databases.

Modules

Some threads call specific portions (or “modules”) of Replication Server to determine the destination of messages and transactions, and to determine what operations to replicate and how to replicate them. Replication Server provides one or more counters for each Replication Server module.

Modules are of three types:

- Multithreaded modules – can have multiple thread instances. They include:
 - Stable Queue Manager (SQM)
 - Stable Queue Transaction (SQT)
 - Distributor (DIST)
 - Data Server Interface (DSI)
 - Replication Server Interface (RSI)
 - Replication Agent (REPAGENT)

- Single-threaded modules – have only one thread instance. All Replication Server daemons, such as the Connection Manager Daemon (dCM), are single-threaded modules.
- Nonthreaded modules – have no thread instance. They include:
 - Message Delivery (MD)
 - Transaction Delivery (TD)
 - Subscription Resolution Engine (SRE)
 - System Table Services (STS)
 - Memory Management (MEM)

Each thread of a multithreaded module has one instance of that module's counters. Single-threaded and nonthreaded modules have only one counter instance.

Counters

Sybase provides over 100 counters for monitoring Replication Server performance. Additional counters may be added in later releases. You can view descriptive information about the current counters using the `rs_helpcounter` stored procedure. See “Viewing information about the counters” on page 605.

Names

Each counter has three different names: a descriptive external name, a display name that you use to identify the counter through RCL, and a unique ID within Replication Server.

Types

Each counter has a type that determines its data sampling method. Counters are of four different sampling types:

- Maximum-value counters – keep only the largest value sampled.
- Average-value counters – keep the total of all values sampled and the number of items sampled.
- Last-value counters – keep the last value of sampled data.
- Cumulative-total counters – keep running total of sampled data.

Status options

Each counter may also have status information that determines calculation operations. Some counters may require additional data processing or information from other counters or modules.

Table 17-1 describes the counter status options. Counters can have zero to several status options.

Table 17-1: Counter status options

Counter status	Description
CNT_INTRUSIVE	Counters that may impact Replication Server performance.
CNT_INTERNAL	Counters used by Replication Server and other counters.
CNT_SYSMON	Counters used by admin statistics, sysmon command.
CNT_MUST_SAMPLE	Counters that sample even if sampling is not enabled.
CNT_NO_RESET	Counters that are not reset after initialization.
CNT_DURATION	Counters that measure duration.
CNT_RATE	Counters that measure rate.
CNT_KEEP_OLD	Counters that keep both the current and previous value.
CNT_CONFIGURE	Counters that keep the run value of a Replication Server configuration parameter.

Enabling sampling and flushing

By default, sampling is enabled and most counters sample data continually with minimal impact on server performance. Counters that may affect performance (intrusive counters) are by default set to off.

When you enable sampling, counters register metrics but do not store them. To store the information, you must also enable flushing. See “Enabling flushing” on page 602.

Most of the parameters described in this section are set at the Replication Server level and affect all counters and/or flushing of all counters. These parameters take effect immediately and do not retain their value after Replication Server shuts down.

However, several module and connection parameters are persistent across Replication Server executions. These parameters include `stats_sampling`, `statss_flush_rssd`, `stats_reset_afterflush`, and `stats_daemon_sleep_time`. Changes to these parameters take effect immediately and do not require you to reboot Replication Server.

Sybase recommends that you set these parameters immediately after loading the software.

❖ **Enabling counter sampling and flushing:**

This is a general procedure for enabling sampling and flushing for both intrusive and non-intrusive sampling. Not all steps are required. If you do not want to enable intrusive counters, for example, skip step 2.

Note This procedure enables sampling and flushing. It provides an easy way to turn sampling and flushing on and off. To turn on flushing for individual threads or modules, see “Configuring modules, connections, and routes” on page 603.

- 1 Enable sampling of non-intrusive counters using the configure replication server set 'stats_sampling' to 'on' command. The default value is “on.”
- 2 Enable sampling of intrusive counters using the admin `stats_intrusive_counters`, 'on' command. The default value is “off.”
- 3 Enable flushing of counters to the RSSD using the configure replication server set 'stats_flush_rssd' to 'on' command. The default value is “off.”
- 4 Enable resetting of counters after flushing using the configure replication server set 'stats_reset_afterflush' to 'on' command. The default value is “on.”
- 5 Set period, in seconds, between flushes with the configure replication server set 'stats_daemon_sleep_time' to *sleep_time* command. The default value is 600 seconds.

Enabling sampling of non-intrusive counters

You enable or disable all sampling at the Replication Server level using the configure replication server command with the `stats_sampling` option. The default is on. The syntax is:

```
configure replication server set 'stats_sampling'
to { 'on' | 'off' }
```

If sampling is disabled, the counters do not record data and no metrics can be flushed to the RSSD.

Enabling sampling of intrusive counters

Most counters sample data with minimal effect on Replication Server performance. Counters that may affect performance—intrusive counters—are enabled separately so that you can enable or disable them without affecting the settings for non-intrusive counters.

Three counters may affect performance:

- `TranTimeLast` – records the time, in one-hundredths of a second, for the DSI thread to process the last transaction in a run.
- `TranTimeMax` – records the maximum time, in one-hundredths of a second, for the DSI thread to process a transaction in a run.
- `TranTimeAve` – records the average time, in one-hundredths of a second, for the DSI thread to process a transaction in a run.

You can enable or disable intrusive counters using the `admin stats_intrusive_counters` command. The default is “off.” The syntax is:

```
admin stats_intrusive_counters, { 'on' | 'off' }
```

Enabling flushing

Use the `configure replication server` command with the `stats_flush_rssd` option to enable or disable flushing. The default is “off.” The syntax is:

```
configure replication server set 'stats_flush_rssd'  
to { 'on' | 'off' }
```

You must enable flushing before you can configure individual modules, connections, and routes to flush.

Enabling reset after flushing

Use the `configure replication server` command with the `stats_reset_afterflush` option to specify that counters are to be reset after flushing. The default is “on.” The syntax is:

```
configure replication server set  
'stats_reset_afterflush' to { 'on' | 'off' }
```

Certain counters, such as rate counters with CNT_NO_RESET status, are never reset. See “Resetting counters” on page 608 for more information.

Setting seconds between flushes

You set the number of seconds between flushes at the Replication Server level using the configure replication server command with the stats_daemon_sleep_time option. The default is 600 seconds. The syntax is:

```
configure replication server set
    'stats_daemon_sleep_time' to sleeptime
```

The minimum value for *sleeptime* is 10 seconds; the maximum value is 3153600 seconds (365 days).

Configuring modules, connections, and routes

A hierarchy of configuration options limit the flushing of counters to the RSSD. admin stats_config_module lets you configure flushing for a particular module or for all modules. For multithreading modules, you can choose to flush metrics from a matrix of available counters. For example, you can configure flushing for a module, for a particular connection, or for all connections.

Configuration parameters that configure counters for flushing are not persistent; they do not retain their values when Replication Server shuts down.

Before you can configure a counter for flushing, make sure that you first enable the sampling and flushing of counters. See “Enabling sampling and flushing” on page 600.

Note Replication Server does not flush counters that have a value of zero.

Flushing for all modules

You can set flushing on for all counters of individual modules or all modules using the admin stats_config_module command. The default is “off.” The syntax is:

```
admin stats_config_module, { module_name |
'all_modules' }, { 'on' | 'off' }
```

module_name is dist, dsi, rsi, sqm, sqt, sts, repagent, cm, or sts.

This command is most useful for single or nonthreaded modules, which have only one thread instance. For multithreaded modules, you have greater control over which threads are set on if you use the `admin stats_config_connection` and `admin stats_config_route` commands.

Note If a module's flushing status is set on, counters for all threads for that module will be set on also.

Flushing for multithreaded modules

The number of threads for a multithreaded module depends on the number of connections and routes Replication Server manages. You can configure flushing for individual threads or groups of threads.

Connections

Use the `admin stats_config_connection` command to enable flushing for threads related to connections. The syntax is:

```
admin stats_config_connection, { data_server, database |
all_connections }, { module_name | all_modules },
[ 'inbound' | 'outbound' ], { 'on' | 'off' }
```

Keyword and parameter descriptions are:

- *data_server* is the name of a data server.
- *database* is the name of a database.
- *all_connections* specifies all database connections.
- *module_name* is dist, dsi, repagent, sqm, or sqt.
- *all_modules* specifies the DIST, DSI, REPAGENT, SQM, and SQT modules.
- *inbound* | *outbound* identifies SQM or SQT for an inbound or outbound queue.

Routes

You can use the `admin stats_config_route` command to save statistics gathered on routes for the SQM or RSI modules. The syntax is:

```
admin stats_config_route, { rep_server |
all_routes }, { module_name | all_modules },
{ 'on' | 'off' }
```

where *rep_server* is the name of the remote Replication Server, *all_routes* specifies all routes from the current Replication Server, and *module_name* is *sqm* or *rsi*.

Note If you configure flushing for a thread, Replication Server also turns on flushing for the module. This does not turn on flushing for existing threads of that module, but all threads will have flushing turned on.

Viewing information about the counters

You can view descriptive information about the counters stored in the *rs_statcounters* table using the *rs_helpcounter* system procedure. See “*rs_helpcounter*” in the *Replication Server Reference Manual* for detailed syntax and usage information.

- To view a list of modules that have counters and the syntax of the *rs_helpcounter* procedure, enter:

```
rs_helpcounter
```

- To view descriptive information about all counters for a specified module, enter:

```
rs_helpcounter module_name
    [, {type | short | long} ]
```

If you enter *type*, Replication Server prints the display name, module name, counter type, and counter status for each of the module’s counters.

If you enter *short*, Replication Server prints the display name, module name, and counter descriptions for each counter.

If you enter *long*, Replication Server prints every column in *rs_statcounters* for each counter.

If you do not enter a second parameter, Replication Server prints the display name, the module name, and the external name of each counter.

- To list all counters that match a keyword, enter:

```
rs_helpcounter keyword [, {type | short | long} ]
```

- To list counters with a specified status, the syntax is:

```
rs_helpcounter { 'intrusive' | 'sysmon' | 'rate'  
                | 'duration' | 'internal' | 'must_sample'  
                | 'no_reset' | 'keep_old' | 'configure' }
```

See Table 17-1 on page 600 for a descriptions of status types.

Viewing current counter values

Replication Server provides a set of admin statistics commands that you can use to display current metrics from enabled counters. Replication Server can display information about these modules: DSI, DSIEXEC, SQT, dCM, DIST, RSI, SQM, REPAGENT, MEM, MD, and MEM_IN_USE.

See “admin statistics” in the *Replication Server Reference Manual* for detailed information about this command.

- To view the current values for one or all the counters for a specified module, enter:

```
admin statistics, module_name [, display_name]
```

where *module-name* is the name of the module and *display_name* is the display name of the counter. To determine the display name of a counter, use `rs_helpcounter`.

- To view current values for all enabled counters, enter:

```
admin statistics, 'all_modules'
```

- To view a summary of values for the DSI, DSIEXEC, REPAGENT, and RSI modules, enter:

```
admin statistics, sysmon [, sample_period]
```

where *sample_period* is the number of seconds for the run. `admin statistics, sysmon [, sample_period]` zeros the counters, samples for the specified sample period, and prints the results. If *sample_period* is 0 (zero) or not present, `admin statistics, sysmon [, sample_period]` prints the current values of the counters.

- To display counter flush status, enter:

```
admin statistics, flush_status
```


Viewing values flushed to the RSSD

You can view information flushed to the `rs_statdetail` and `rs_statrun` tables using `select` and other Transact-SQL commands. If, for example, you want to display flushed information from the dCM module counters, you might enter:

```
select counter_name, module_name, instance_id,
       counter_val, run_date from rs_statcounters c,
       rs_statdetail d, rs_statrun r
where c.counter_id = d.counter_id and
      d.run_id = r.run_id and c.module_name = 'CM'
order by counter_name
```

In this instance, the counters have been configured to flush every two minutes using the `configure replication server set 'stats_daemon_sleep_time' to '120'` command.

Replication Server displays:

counter_name	module_name	instance_id	counter_val	run_date
CM:Daemon dSTATS schedule count	CM	13	1	Feb 26 2001 11:16AM
CM:Daemon dSTATS schedule count	CM	13	2	Feb 26 2001 11:18AM
CM:Daemon dSTATS schedule count	CM	13	3	Feb 26 2001 11:20AM
CM:Daemon dSTATS schedule count	CM	13	4	Feb 26 2001 11:22AM
CM:Daemon dSTATS schedule count	CM	13	5	Feb 26 2001 11:24AM
CM:Daemon dSUB schedule count	CM	13	1	Feb 26 2001 11:16AM
CM:Daemon dSUB schedule count	CM	13	2	Feb 26 2001 11:18AM
CM:Daemon dSUB schedule count	CM	13	3	Feb 26 2001 11:20AM
CM:Daemon dSUB schedule count	CM	13	4	Feb 26 2001 11:22AM
CM:Daemon dSUB schedule count	CM	13	5	Feb 26 2001 11:24AM

Resetting counters

Counters are reset when a thread starts. In addition, some counters are reset automatically at the beginning of a sampling period.

You can reset counters by:

- Configuring Replication Server to ensure that counters are reset after sampling data is flushed to the RSSD. Use the `configure replication server set 'stats_reset_afterflush' to 'on'` command.
- Issuing the `admin statistics, reset` command to reset all counters.

You can reset all counters except counters with `CNT_NO_RESET` status, such as rate counters, which are never reset. Counters that can be reset, reset to zero.

dSTATS daemon thread

The dSTATS daemon thread supports Replication Server's counters feature by:

- Managing the interface for flushing counters to the RSSD.
- Calculating derived values when the daemon thread wakes up.

dSTATS manages the interface when Replication Server has been configured to flush statistics to the RSSD using the `configure replication server` command and the `stats_flush_rssd` parameter.

You can configure a sleep time for dSTATS using the `configure replication server` command and the `stats_daemon_sleep_time` parameter. When the daemon wakes up, it attempts to calculate derived statistics such as the number of DSI-thread transactions per second or the number of RepAgent bytes delivered per second.

Handling Errors and Exceptions

This chapter describes various error handling methods for Replication Server.

Name	Page
General error handling	609
Error log files	610
Data server error handling	615
Exceptions handling	620
DSI duplicate detection	625
Duplicate detection for system transactions	626

Refer to the *Replication Server Troubleshooting Guide* for information about resolving specific errors.

General error handling

Replication Server passes messages to data servers and other Replication Servers while they are accessible and queues messages when connections are down. Using Sybase Central, you can monitor the status of the replication system and troubleshoot problems as they arise.

Normally, short-term failures of networks and data servers do not require special error handling or intervention. When the failure is corrected, replication system components resume their work automatically. Lengthier failures may require intervention if there is not enough disk space to queue up messages or if it is necessary to reconfigure the replication system to work around the failure.

Failures of some system components, such as Replication Server partitions or primary databases, also require user intervention. Refer to Chapter 19, “Replication System Recovery” for more information about recovery procedures.

A Replication Server's response to errors depends on the kind of error, source of the error, and how the Replication Server is configured. Replication Server handles errors in these ways:

- Logs errors in its error log file.
- Responds to data server errors based on configuration settings.
- If transactions fail to commit in a database, writes the transactions to the exceptions log for manual resolution.
- Detects duplicate transactions after system restart.

Error log files

This section describes error log files in the replication system. You can access log files to help you troubleshoot Replication Server and RepAgent. To view skipped transactions that are written to system tables, you can access the Adaptive Server for the Replication Server managing a specified database. Refer to the *Replication Server Troubleshooting Guide* for details on troubleshooting errors.

Replication Server allows user-definable error processing in response to data server errors. For details, see “Data server error handling” on page 615.

RSM

For instructions on viewing and refreshing the error log in Sybase Central, see “Viewing the Error Log” in Replication Server's plug-in help.

Replication Server error log

The Replication Server error log is a text file where Replication Server writes informational and error messages.

By default, the Replication Server error log file name is *repserver.log*, and resides in the directory where you started the Replication Server. You can specify the name and location of the error log file by using the `-E` command line flag when you start the Replication Server or in a Replication Server run file

Each log message begins with a letter to indicate the message type. Table 18-1 lists the possible message types.

Table 18-1: Message types in the Replication Server error log

Error code	Description
I	An informational message.
W	A warning about a condition that has not yet caused an error, but may require attention. An example is running out of a resource.
E	An error that does not prevent further processing, such as a site that is unavailable.
H	A Replication Server thread has died. An example is a lost network connection.
F	Fatal. A serious error caused Replication Server to exit. An example is starting the Replication Server with an incorrect configuration.
N	Internal error. These errors are caused by anomalies in the Replication Server software. Report these errors to Sybase Technical Support.

Informational messages

For informational messages in the error log, the format is as follows:

```
I. date: message
```

The letter “I” at the beginning of a message means that the message is provided for information. It does not mean that an error occurred. For example, Replication Server outputs the following messages as it drops a subscription:

```
I. 95/11/01 05:41:54. REPLICATE RS: Dropping
subscription <authors_sub> for replication definition
<authors> with replicate at <SYDNEY_DS.pubs2>
I. 95/11/01 05:42:02. SQM starting: 104:-2147483527
authors.authors_sub
I. 95/11/01 05:42:12. SQM Stopping: 104:-2147483527
authors.authors_sub
I. 95/11/01 05:42:20. REPLICATE RS: Dropped
subscription <authors_sub> for replication definition
<authors> with replicate at <SYDNEY_DS.pubs2>
```

Error and warning messages

For messages other than informational messages, the format is as follows:

```
severity, date. ERROR #error_number
thread_name(context) - source_file(line) message
```

If the message is a warning, “ERROR” in the above format becomes “WARNING.”

The *severity* is either W, E, H, F, or N, as listed in Table 18-1. The *date* is the date and time that the error occurred. The *error_number* is the Replication Server error number.

The *thread_name* is the name of the Replication Server thread that received the error. See Chapter 2, “Replication Server Technical Overview” and Chapter 16, “Performance Tuning” for details about Replication Server threads. The *context* provides some information about the thread’s context at the time the error occurred.

The *source_file* and *line* point to the program file and line number in the Replication Server source code where the error was reported.

The *message* is the full text of a message from a Replication Server. It is in the language specified in the *RS_language* configuration parameter. Some messages also include a message from a data server, or one of the component libraries that Replication Server uses.

Note Replication Server puts question marks (?) in messages where more specific information is not available. For example, if an error occurs during initialization, Replication Server may not yet have completed some internal structures, so it prints question marks in place of information it has not yet collected.

The following example shows the Replication Server error log entry for a data server:

```
E. 95/11/01 05:30:52. ERROR #1028 DSI(SYDNEY_DS.pubs2)
- dsigmint.c(3522)Message from server:
  Message: 2812, State: 4, Severity: 16 --
  'Stored procedure 'upd_authors' not found.
H. 95/11/01 05:30:53. THREAD FATAL ERROR #5049
DSI(SYDNEY_DS.pubs2) - dsigmint.c(3529)
  The DSI thread for database 'SYDNEY_DS.pubs2' is being
  shutdown because of error action mapped from data server
  error '2812'. The error was caused by output command '1'
  mapped from source command '2' of the transaction.
```

The messages indicate that Adaptive Server returned error number 2812, causing Replication Server to take the `stop_replication` action. See “Assigning actions to data server errors” on page 618.

Finding the name of the Replication Server error log

Use the `admin log_name` command to find the name of the current Replication Server error log file. Replication Server displays the path to the log file, as the following UNIX example shows:

```
Log File Name
-----
/work/sybase/SYDNEY_RS/SYDNEY_RS.log
```

Changing to a new Replication Server log file

To begin a new error log file, use the `admin set_log_name` command. This command closes the current log file and opens a new one. Subsequent messages are written in the new log file.

Following is an example of the `admin set_log_name` command for UNIX:

```
admin set_log_name,
'/work/sybase/SYDNEY_RS/951101.log'
```

The previous log remains active if Replication Server fails to create and open the new log file.

RepAgent error log messages

All RepAgent error, trace, and information messages are logged in the Adaptive Server error log file. Each message identifies the RepAgent that logged the error in the string “RepAgent (*dbid*)”, which appears in the first line of the message. *dbid* is the database identification number of the RepAgent that logged the error.

Here is an example of an information message:

```
RepAgent(dbid): Recovery of transaction log is
complete. Please load the next transaction log dump and
then start up the Rep Agent Thread with
sp_start_rep_agent, with 'for recovery' specified.
```

The Adaptive Server error log is a text file. The messages are printed in the language specified at Adaptive Server. RepAgent records errors and informational messages that occur when transferring replicated objects from the Adaptive Server transaction log and converting them into commands. RepAgent errors are generally in the 9200 to 9299 range.

Adaptive Server performs actions based on the severity and recoverability of an error. Some errors are for information only, others cause Adaptive Server to retry the operation that caused the error until it succeeds, and still others indicate an error too severe to continue and RepAgent shuts down. For more information about the Adaptive Server error log file, refer to the *Adaptive Server System Administration Guide*.

For information about LTM error messages, see Appendix B, “LTM for SQL Server.”

Sample error messages

This section describes some common RepAgent error messages and possible solutions.

- In this example, the RepAgent login name is not present on the Replication Server.

```
RepAgent(6): Failed to connect to Replication
Server. Please check the Replication Server,
username, and password specified to
sp_config_rep_agent. RepSvr = repserver_name, user =
RepAgent_username
RepAgent(6): This Rep Agent Thread is aborting due
to an unrecoverable communications or Replication
Server error.
```

You must either add RepAgent’s login name to Replication Server or change RepAgent’s login name.

- In this example, RepAgent cannot connect to Replication Server.

```
RepAgent(7): The Rep Agent Thread will retry the
connection to the Replication Server every 60
second(s). (RepSvr = repserver_name.)
```

Check Replication Server status. If Replication Server is down, resolve the problem and restart. Otherwise, wait for possible network problem to resolve.

Data server error handling

Replication Server allows user-definable error processing for data server errors. This is accomplished by creating an error class for a database and specifying responses for each error that the data server returns when the error is encountered in the database. The data server returns the defined errors to Replication Server. Table 18-2 lists the RCL commands and Adaptive Server system procedures that manage errors and error classes.

Table 18-2: RCL commands and system procedures for error processing

Command	Description
rs_helpclass	Adaptive Server system procedure that displays the name of each existing error class, function-string class, and their primary Replication Server
create error class	Creates a new error class
drop error class	Drops an existing error class
assign action	Specifies an error processing action for one or more data server errors
create connection	Associates an error class with a new database connection
alter connection	Associates an error class with an existing database connection

A default error class, `rs_sqlserver_error_class`, is provided for Adaptive Server databases.

Creating an error class

You can define a single error class to use with all databases managed by the same type of data server. For example, you can use the default Adaptive Server error class, `rs_sqlserver_error_class`, with any Adaptive Server database. There is no need to create another error class unless a database has special error-handling requirements.

An error class is a name used to group error action assignments. The syntax for the create error class command is:

```
create error class error_class
```

For example, to create an error class named `pubs2_error_class`, use this command:

```
create error class pubs2_error_class
```

Initially, `rs_sqlserver_error_class`, the default error class that is predefined to work with Adaptive Server databases, does not have a primary site. Since you can only create server-wide error classes at a primary site for a class, you need to designate one of the Replication Servers as a primary site for a Adaptive Server error class.

You must specify a primary site before you can modify a default error class. You can designate a site as primary by executing the `create error class` command for a Adaptive Server error class at that site. To do this, execute `create error class rs_sqlserver_error_class` at the primary site. Make sure all other Replication Servers have direct or indirect routes from the primary site.

The default error action for all errors returned by a data server is `stop_replication`. This is also the most serious action: it suspends replication for the database, as if you entered the `suspend connection` command. To assign less severe actions to errors you want to handle differently, use the `assign action` command. See “Assigning actions to data server errors” on page 618 for more information.

Initializing a new error class

After you have created a new error class, you can initialize it with error actions from an error class such as the system-provided `rs_sqlserver_error_class`. To do this, use the `rs_init_erroractions` stored procedure:

```
rs_init_erroractions new_error_class, template_class
```

For example, to create the error class `pubs2_error_class`, based on the template error class `rs_sqlserver_error_class`, enter:

```
rs_init_erroractions pubs2_error_class,  
rs_sqlserver_error_class
```

Then use the `assign action` command to change the actions for individual errors.

Dropping an error class

The `drop error class` command drops an error class and all actions associated with it. The error class must not be in use with an active database connection when you drop it. The syntax for `drop error class` is:

```
drop error class error_class
```

For example, to drop the `pubs2_error_class` error class, use this command:

```
drop error class pubs2_error_class
```

You cannot drop the `rs_sqlserver_error_class` error class.

Changing the primary Replication Server for an error class

Use the `move primary` command to change the primary site for an error class. This is necessary when you are changing the primary site from one Replication Server to another so that error actions can be distributed through new routes. For example, you must use this command if you are dropping from the replication system the Replication Server that is the current primary site for an error class.

Before you execute `move primary`, make sure that a route exists from:

- The new primary site to each Replication Server that will use the error class
- The current primary to the new primary site
- The new primary to the current primary site

The syntax for the `move primary` command, for error classes, is:

```
move primary of error class class_name  
to replication_server
```

Execute the `move primary` command at the Replication Server that you want to designate as the new primary site for the error class.

- *class_name* – the name of the error class whose primary Replication Server is to be changed.
- *replication_server* – specifies the new primary Replication Server for the error class.

The following command changes the primary site for the `pubs2_error_class` error class to the `TOKYO_RS` Replication Server where the command is entered:

```
move primary of error class pubs2_error_class  
to TOKYO_RS
```

For the default error class, `rs_sqlserver_error_class`, no Replication Server is the primary site until you assign one as the primary site. You must specify a primary site before you can use the `assign action` command to change default error actions.

To specify a primary site for the default error class, execute the following command in that Replication Server:

```
create error class rs_sqlserver_error_class
```

After you have executed this command, you can use the `move primary` command to change the primary site for the error class.

Displaying error class information

The Adaptive Server `rs_helpclass` stored procedure displays the names of existing error classes and function-string classes in the replication system and their primary Replication Servers. For example:

```
rs_helpclass error_class
Error Class(es)                PRS for class
-----
rs_sqlserver_error_class      Not Yet Defined
```

Refer to “`rs_helpclass`” in Chapter 6, “Adaptive Server Stored Procedures,” in the *Replication Server Reference Manual* for more information.

Assigning actions to data server errors

The `assign action` command specifies the action to take for errors that a data server can return to Replication Server. The syntax for the `assign action` command is:

```
assign action {ignore | warn | retry_log | log |
retry_stop | stop_replication}
for error_class
to data_server_error [, data_server_error]...
```

For example, to instruct Replication Server to ignore Adaptive Server errors 5701 and 5703:

```
assign action ignore
for rs_sqlserver_error_class
to 5701, 5703
```

The default error class provided for Adaptive Server databases is `rs_sqlserver_error_class`. You must create this error class at a primary site before you can use the `assign action` command to change default error actions. The `data_server_error` parameter is the data server error number.

Enter one of the six possible error actions at the Replication Server where the error class was created. These actions are listed in Table 18-3, in order of severity: ignore is the least severe action and stop_replication is the most severe.

When a transaction causes multiple errors, Replication Server chooses just one action—the most severe action assigned to any of the errors that occurred. To return an error to the default error action, stop_replication, you must reassign it explicitly.

Table 18-3: Replication Server actions for data server errors

Action	Description
ignore	Assume that the command succeeded and that there is no error or warning condition to process. This action can be used for a return status that indicates successful execution.
warn	Log a warning message, but do not roll back the transaction or interrupt execution.
retry_log	Roll back the transaction and retry it. The number of retry attempts is set with the configure connection command. If the error continues after retrying, write the transaction into the exceptions log, and continue, executing the next transaction.
log	Roll back the current transaction and log it in the exceptions log; then continue, executing the next transaction.
retry_stop	Roll back the transaction and retry it. The number of retry attempts is set with the configure connection command. If the error recurs after retrying, suspend replication for the database.
stop_replication	Roll back the current transaction and suspend replication for the database. This is equivalent to using the suspend connection command. This action is the default. Since this action stops all replication activity for the database, it is important to identify the data server errors that can be handled without shutting down the database connection, and assign them to another action.

Displaying assigned actions for error numbers

Execute the rs_helperror stored procedure to display the action assigned for an error number. The syntax for the rs_helperror stored procedure is:

```
rs_helperror server_error_number [, v]
```

where *server_error_number* parameter is the data server error number of the error you want information for. The *v* parameter specifies “verbose” reporting. When you supply this option, rs_helperror also displays the Adaptive Server error message text, if available. Refer to “rs_helperror” in Chapter 6, “Adaptive Server Stored Procedures,” in the *Replication Server Reference Manual* for more details.

Exceptions handling

When a transaction submitted by Replication Server fails, Replication Server records the transaction in the exceptions log in the RSSD. The Replication System Administrator at the site must resolve the transactions in the exceptions log. See “Accessing the exceptions log” on page 622.

Transactions can fail due to errors such as duplicate keys, column value checks, and insufficient disk space. They may also be rejected for reasons such as insufficient permissions, version control conflicts, and invalid object references.

Because skipping a transaction causes inconsistency and can have an adverse affect on the system, you should review on a regular basis any transactions that have been recorded in the exceptions log and resolve them. The best resolution for a transaction may depend on the client application that originated it. For example, if a failed transaction corresponds to a real-world event, such as a cash withdrawal, the transaction must somehow be applied.

Refer to the *Replication Server Troubleshooting Guide* for more information on the implications of skipping a transaction.

RSM

For information about exceptions handling, correcting transactions, and purging the exceptions log in Sybase Central, see “Managing the Exceptions Log” in Replication Server’s plug-in help.

Handling failed transactions

This section outlines the recommended process for handling failed transactions that require manual intervention.

Suspend database connection

When a data server begins rejecting transactions because of a temporary failure, such as lack of space in a database or log file, you can suspend the database connection until the error is corrected.

If the database connection is not suspended, Replication Server writes the transactions into the exceptions log for the database. Since these transactions must then be resolved manually, you can save time by shutting down the connection until the error condition is corrected.

While a database connection is suspended, Replication Server stores transactions in a stable queue. When the connection is resumed, the stored transactions are sent to the data server.

To stop the flow of transactions from a Replication Server to a database, use the suspend connection command:

```
suspend connection to data_server.database
```

The command requires sa permission and must be entered at the Replication Server that manages the database.

Analyze and resolve the problem

You then need to determine why the transaction failed, make corrections or adjustments, and resubmit the transaction. For example, if a transaction failed because the maintenance user had insufficient permissions, grant the maintenance user the needed permissions and retry the transaction.

If you are resolving transactions in the exceptions log:

- 1 Retrieve a list of the transactions from the exceptions log. See “Accessing the exceptions log” on page 622.
- 2 Investigate the transactions to determine the cause of failure and the best method for resolution.
- 3 Resolve the transactions according to your plan. For example, you might correct a permissions problem and then resubmit a transaction.
- 4 Delete resolved transactions from the exceptions log. See “Deleting transactions from the exceptions log” on page 624.

Resume the connection

Restart the flow of transactions for a suspended database connection with the resume connection command. The same command is used whether you suspended the connection intentionally, using the suspend connection command, or whether it was suspended by Replication Server as the result of an error action. The syntax for resume connection is:

```
resume connection to data_server.database  
[skip transaction]
```

The command requires sa permission and must be entered at the Replication Server that manages the database.

Use the skip transaction clause to instruct Replication Server to ignore the first transaction in the queue. You may need to do this if a transaction continues to fail each time you resume the connection.

Accessing the exceptions log

The exceptions log is implemented in three system tables in the RSSD: `rs_exceptshdr`, `rs_exceptscmd`, and `rs_systext`.

- `rs_exceptshdr` stores general information about a transaction in the exceptions log, such as the transaction ID, the database where the transaction originated, and the data server where the error occurred.
- `rs_exceptscmd` stores information needed to retrieve the source and output command text from `rs_systext`. The source is the text of the transaction that Replication Server receives from RepAgent. The output command is the result of Replication Server function string mapping. There is one row for each source and output command.
- `rs_systext` holds the text of the source and output commands. Each command may have multiple rows in `rs_systext`, indexed by the sequence column.

Refer to Chapter 8, “Replication Server System Tables,” in the *Replication Server Reference Manual* for a description of each of the columns in these system tables.

Displaying transactions in the exceptions log

You can display a summary list of all transactions in the exceptions log using the `rs_helpexception` stored procedure. The syntax for the `rs_helpexception` stored procedure is:

```
rs_helpexception [transaction_id, [, v]]
```

If you supply a valid *transaction_id* and *v* for “verbose” reporting, `rs_helpexception` displays a detailed description of a transaction. Use `rs_helpexception` with no parameters to obtain *transaction_id* numbers for all transactions in the exceptions log.

Querying the exceptions log system tables

You can join the `rs_exceptshdr` and `rs_exceptscmd` system tables on the `sys_trans_id` column.

You can also join the `rs_exceptscmd` and `rs_systext` system tables to retrieve the text of a transaction. To do this, join the `cmd_id` column in `rs_exceptscmd` to the `parentid` column in `rs_systext`.

Figure 18-1 illustrates the exceptions log system tables.

Figure 18-1: Exceptions log system tables

rs_exceptshdr

sys_trans_id	rs_id
rs_trans_id	binary
app_trans_name	varchar
orig_siteid	int
orig_site	varchar
orig_db	varchar
orig_time	datetime
orig_user	varchar
error_siteid	int
error_site	varchar
error_db	varchar
log_time	datetime
ds_error	int
ds_errmsg	varchar
error_src_line	int
error_proc	int
err_output_line	int
log_reason	char
trans_status	smallint
retry_status	smallint
app_usr	varchar
app_pwd	varchar

rs_exceptscmd

sys_trans_id	rs_id
src_cmd_line	int
output_cmd_index	int
cmd_type	char
cmd_id	rs_id

rs_systext

prsid	int
parentid	rs_id
texttype	char
sequence	int
textval	varchar

The `rs_exceptshdr` system table contains descriptive information about the transactions in the exceptions log, including the following:

- User-assigned transaction name

- Site and database where the transaction originated
- User at the origin site who submitted the transaction
- Information about the error that caused the transaction to be recorded in the exceptions log

To retrieve a list of the excepted transactions for a given database, use, for example, the following query:

```
select * from rs_exceptshdr
  where error_site = 'data_server'
  and error_db = 'database'
  order by log_time
```

To retrieve the source and output text for a transaction with a given system transaction ID, use:

```
select t.texttype, t.sequence,
  t.textval
  from rs_systext t, rs_exceptscmd e
  where e.sys_trans_id = sys_trans_id
  and t.parentid = e.cmd_id
  order by e.src_cmd_line, e.output_cmd_index,
  t.sequence
```

Refer to Chapter 8, “Replication Server System Tables,” in the *Replication Server Reference Manual* for a list of all of the columns in these Replication Server system tables.

Deleting transactions from the exceptions log

To delete a transaction from the exceptions log, use the `rs_delexception` stored procedure.

```
rs_delexception [transaction_id]
```

With no parameters, `rs_delexception` displays a summary of transactions in the exceptions log. If you supply a valid *transaction_id*, `rs_delexception` deletes a transaction. You can find the *transaction_id* for a transaction by using either `rs_helpexception` or `rs_delexception` with no parameters.

DSI duplicate detection

The DSI records the last transaction committed or written into the exceptions log so that it can detect duplicates after a system restart. Each transaction is identified by a unique origin database ID and an origin queue ID that increases for each transaction.

The last transaction committed from each origin database is recorded at a data server by executing the function strings defined for the data server's function-string class. For the system-defined classes, this is done in the function string for a commit command, that is, the `rs_commit` function. Every function-string class supports the `rs_get_lastcommit` function, which returns the `origin_qid` and `secondary_qid` for each origin database. The `secondary_qid` is the ID of the queue used for subscription materialization or dematerialization.

The `origin_qid` and `secondary_qid` for the last transaction written into the exceptions log from each origin is recorded into the `rs_exceptslast` system table. However, transactions logged explicitly by the `sysadmin log_first_tran` command are not recorded in this system table. These transactions are logged, but they are not skipped.

When a DSI is started or restarted, it gets the `origin_qid` returned by the `rs_get_lastcommit` function and the one stored in the `rs_exceptslast` system table. It assumes that any transaction in the queue with an `origin_qid` less than the larger of these two values is a duplicate and ignores it.

If the `origin_qid` values stored in a data server or the `rs_exceptslast` system table are modified by mistake, non-duplicate transactions may be ignored or duplicate transactions may be reapplied. If you suspect that this is happening in your system, check the values stored and compare them with the transactions in the database's stable queue to determine the validity of the values. If the values are wrong, you must modify them directly.

Refer to the *Replication Server Troubleshooting Guide* for details on how to dump transactions in a queue.

Duplicate detection for system transactions

truncate table and certain supported DDL commands are not logged, although they can be replicated to standby and replicate databases. Refer to “Supported DDL commands and system procedures” on page 488 for a list of DDL commands supported for replication. Refer to the *Adaptive Server Reference Manual* for information about each DDL command.

Replication Server copies these commands as system transactions, in which Replication Server “sandwiches” the truncate table or similar command between two complete transactions. Execution of the first transaction is recorded in the replicate database in the secondary_qid column of the rs_lastcommit table and in the origin_qid column of that table. If Replication Server records the second transaction, the system transaction has completed, and Replication Server clears the secondary_qid column.

If there is a system failure, and you see the following error message when the system restarts:

```
5152 DSI_SYSTRAN_SHUTDOWN,"There is a system
transaction whose state is not known. DSI will be
shutdown."
```

a system command has not completed, and the connection shuts down. You must verify whether the command within the system transaction has executed at the replicate database.

- If the command has executed, or if you choose to execute the command yourself, you can skip the first transaction in the queue and continue with the second transaction when you resume the connection. At the replicate Replication Server, enter:

```
resume connection to data_server.database
skip transaction
```

- If the command has not executed, you can fix the problem and then execute the first command in the queue. At the replicate Replication Server, enter:

```
resume connection to data_server.database
execute transaction
```

You *must* include the skip transaction or execute transaction clause with resume connection. Otherwise, Replication Server does not reset the secondary_qid correctly, and the error message reappears.

Replication System Recovery

This chapter describes how to prevent or recover from certain kinds of system failures in a replication system.

Name	Page
How to use recovery procedures	628
Configuring the replication system to support Sybase Failover	628
Configuring the replication system to prevent data loss	633
Recovering from partition loss or failure	637
Recovering from truncated primary database logs	642
Recovering from primary database failures	645
Recovering from RSSD failure	648
Recovery support tasks	663

While Replication Server tolerates most failure conditions and recovers from them automatically, some failures require user intervention. This chapter identifies those failures and provides procedures for recovery. These procedures are designed to maintain the integrity of the replication system by recovering lost and corrupted data and restoring that data to its previous state.

You should design, install, and administer your replication system with backup and recovery in mind. We assume that dumps are performed on a regular basis and that appropriate tools and settings for handling recovery are in place. See “Creating coordinated dumps” on page 636 for details on performing dumps.

In this chapter, the **current** Replication Server refers to the one with a database (for example, RSSD) that you are recovering. An **upstream** Replication Server has a direct or indirect route to the current Replication Server. A **downstream** Replication Server is one to which the current Replication Server has a direct or indirect route.

How to use recovery procedures

When using recovery procedures in this chapter, always write down or check off recovery steps as you perform them. Such information can help Sybase Technical Support determine where you are in the recovery procedure, if necessary.

Table 19-1 lists failure conditions described in this chapter, and indicates where to find information on corresponding failure symptoms and recovery procedures.

Table 19-1: Overview of available recovery procedures

Failure condition	For symptoms and recovery procedures
Replication Server partition loss or failure	“Recovering from partition loss or failure” on page 637
Truncated primary database logs	“Recovering from truncated primary database logs” on page 642
Primary database failure	“Recovering from primary database failures” on page 645
RSSD failure	“Recovering from RSSD failure” on page 648

Recovery procedures are *only* intended for the specific situations noted in this chapter. Do not use recovery procedures for replication system problems such as failure to replicate data.

Warning! Use recovery procedures in this chapter only for the failure condition specific to the procedure. Attempting to use recovery procedures on conditions other than those specified can complicate your problem and require more drastic recovery actions.

Refer to the *Replication Server Troubleshooting Guide* for help in diagnosing and correcting problems.

Configuring the replication system to support Sybase Failover

This section describes how Replication Server version 12.0 and later supports Sybase Failover available in Adaptive Server Enterprise version 12.0 and later.

Overview

Sybase Failover allows you to configure two version 12.0 and later Adaptive Servers as companions. If the primary companion Adaptive Server fails, that server's devices, databases, and connections can be taken over by the secondary companion Adaptive Server.

You can configure a high availability system either asymmetrically or symmetrically.

An *asymmetric configuration* includes two Adaptive Servers that are physically located on different machines, but share the same system devices, system/master databases, user databases, and user logins. These two servers are connected so that if one of the servers is brought down, the other assumes its workload. The secondary Adaptive Server acts as a “hot standby” and does not perform any work until failover occurs.

A *symmetric configuration* also includes two Adaptive Servers running on separate machines, but each Adaptive Server is fully functional with its own system devices, system/master databases, user databases, and user logins. If failover occurs, either Adaptive Server can act as a primary or secondary companion for the other Adaptive Server.

In either setup, the two machines are configured for dual access, which makes the disks visible and accessible to both servers.

In a replication system, where Replication Server makes many connections to Adaptive Servers, you can enable or disable Failover support of the database connections initiated by a Replication Server to Adaptive Servers. When you enable Failover support, Replication Servers connected to an Adaptive Server that fails are automatically switched to the second companion machine, reestablishing network connections.

See the Adaptive Server Enterprise documentation for more detailed information about how Sybase Failover works in Adaptive Server. See the Appendix, “High Availability on Sun Cluster 2.2,” for information about Failover support for Replication Server.

Enabling Failover support in Replication Server

You enable Failover support for each Replication Server in your system; once for the RSSD connection, and once for all other database connections from the specified Replication Server to Adaptive Servers.

You cannot enable Failover support for individual connections, except the RSSD connection.

The default for Failover support in Replication Server is “off” for all connections from a Replication Server to Adaptive Servers.

For continuing replication, you should enable Failover support for all connections. However, in some cases you may want to disable connection Failover when the secondary server’s workload exceeds its capacity.

How Sybase Failover works with Replication Server

To configure Sybase Failover from Replication Server to Adaptive Server, the Adaptive Server must be configured to allow connection failover. You can configure Sybase Failover for a Replication Server from RSM using the Server Configuration dialog box.

When Adaptive Servers are in failover companion mode and the primary companion fails, the secondary companion takes over the workload. Incomplete transactions or operations that require updates to the RSSD fail. Replication Server retries existing connections, but new connections are failed over.

For Data Server Interface (DSI) connections, the DSI retries failed transactions after a brief sleep.

For RSSD connections, user commands that are executed during failover do not succeed. Internal operations (such as updates to locator, disk segment, and so on) should not fail. Replication of RSSD objects should be covered by the DSI.

Asynchronous commands (for example, subscription, routing, and standby commands) may be rejected or encounter errors and require recovery if the commands have been accepted but not completed. For example, a create subscription command may have been accepted, but the subscription may still be being created.

Note Failover support is not a substitute for warm standby. While warm standby keeps a copy of a database, Failover support accesses the same database from a different machine. Failover support works the same for connections from Replication Server to warm standby databases.

Requirements

- To enable Failover support, a Replication Server must connect to Adaptive Servers that are version 12.0 or later and configured for Failover.
- Failover of Replication Server System Databases (RSSDs) and user databases is configured directly through the Adaptive Server.
- Failover support responds only to failover of the Adaptive Servers; that is, failover of Replication Servers is not supported.
- LTMs do not support Failover; that is, RepAgents are required.
- Adaptive Server is responsible for the RepAgent thread failover and its reconnection to Replication Server after failover/failback.
- Each Replication Server configures its own connections.

Enabling Failover support for an RSSD connection

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 Chapter 2, “Configuring Replication Server and Adding New Databases,” in the *Replication Server Configuration Guide* for your platform.
- Edit the Replication Server’s configuration file after you have installed the Replication Server.
 - a Use a text editor to open the Replication Server’s configuration file. The default file name is the Replication Server name with a “.cfg” extension.

The configuration file contains one line per entry.

- b Find the line “RSSD_ha_failover=no” and change it to:

```
RSSD_ha_failover=yes
```

- c 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.

Enabling Failover support for non-RSSD database connections

You can enable Failover support for new database connections from the Replication Server or RSM Server to Adaptive Servers using either of the procedures in this section.

For more information about Sybase Failover, refer to the Adaptive Server Enterprise book *Using Sybase Failover in a High Availability System*.

❖ Enabling Failover support using the Replication Server Manager (RSM) plug-in for Sybase Central

- 1 Right-click the Replication Server or the RSM Server icon in the Sybase Central view for which you want to configure Sybase Failover.
- 2 Select Configure from the drop-down menu. The Server Configuration dialog box opens.
- 3 Choose General from the Categories drop-down list.
- 4 Select HA_failover from the list of configuration parameters. You may need to scroll down through the list to find the Sybase Failover parameter.
- 5 Enter the correct value in the Pending Value column.

For Replication Server:

- Enter “on” to turn on Sybase Failover.
- Enter “off” to turn off Sybase Failover.

For RSM Server:

- Enter “1” to turn on Sybase Failover.
- Enter “0” to turn off Sybase Failover.

- 6 Click OK. The Server Configuration dialog box closes.
- 7 Restart Replication Server or RSM Server, respectively, to have the new parameter value take effect.

See “Starting Replication Server” in Chapter 4 of the *Replication Server Administration Guide* for instructions.

❖ Enabling Failover support using *configure replication server*

- 1 If necessary, start the Replication Server, as described in the section “Starting Replication Server” in Chapter 4 of the Replication Server Administration Guide.
- 2 Log in to the Replication Server:

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

where *user_name* must have Administrator privileges. Specify the name of the Replication Server using the -S flag.

When your login is accepted, isql displays a prompt:

```
1>
```

- 3 Enter the following RCL command:

```
configure replication server  
set ha_failover to 'on'
```

Configuring the replication system to prevent data loss

This section contains recommended measures for preventing data loss in the event of an irrecoverable database error. If used properly, these measures allow you to restore replicated data using the system recovery procedures.

Save interval for recovery

Replication Servers are designed to store messages from their source and forward them to their destinations. To increase the chances of recovering online messages after rebuilding stable queues, you can set save intervals, measured in minutes, for routes between Replication Servers. A save interval is the amount of time that a message is stored after it has been forwarded. You can also set save intervals for a physical or logical database connection from a Replication Server, allowing Replication Server to save messages in a DSI outbound queue.

To find the current save interval for a route or connection, use the `admin who, sqm` command. The `Save_Int:Seg` column holds two values. The value preceding the colon is the save interval. The value after the colon is the first saved segment in the stable queue.

Details on setting save intervals for routes and connections are described in the following sections.

Routes between Replication Servers

If the Replication Server has suspended routes, or if a network or data server connection is down, a backlog of messages may accumulate in the Replication Server's stable queues. The chance of recovering these messages decreases with time. Source Replication Servers may already have deleted messages from their stable queues and database logs may already have been truncated.

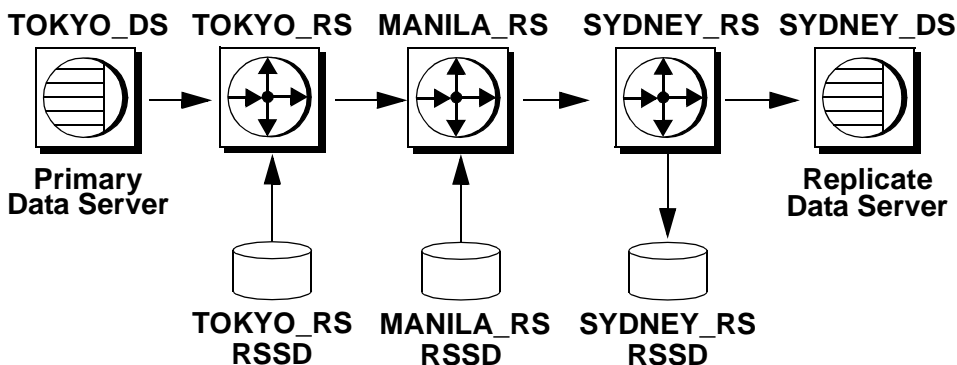
When you set the *save_interval* for each route between Replication Servers, you allow each Replication Server to retain messages for a minimum period of time after the next site in the route acknowledges that it has received the messages. The availability of these messages increases the chance of recovering online messages after queues are rebuilt.

For example, in Figure 19-1 on page 634, Replication Server TOKYO_RS maintains a direct route to MANILA_RS, and MANILA_RS maintains a direct route to SYDNEY_RS.

TOKYO_RS retains messages for a period of time after MANILA_RS has received them. If MANILA_RS experiences a partition failure, it requires that TOKYO_RS to resend the backlogged messages. MANILA_RS can also retain messages to allow SYDNEY_RS to recover from failures.

When all of the messages stored on a stable queue segment are at least as old as the *save_interval* setting, Replication Server deletes the segment so it can be reused.

Figure 19-1: Save interval example



Setting the save interval for routes

To set the *save_interval* for a route, execute the alter route command at the source Replication Server. Using as an example the replication system in Figure 19-1, here is the command to set Replication Server TOKYO_RS to save for one hour any messages destined for MANILA_RS:

```
alter route to MANILA_RS
set save_interval to '60'
```

By default, the *save_interval* is set to 0 (minutes). For systems with low volume, this may be an acceptable setting for recovery, since Replication Server does not delete messages immediately after receiving acknowledgment from destination servers. Rather, messages are deleted periodically in large chunks.

However, to accommodate the volume and activity of sites that receive distributions from the Replication Server and to increase the chance of full recovery from database or partition failures, you may want to change the *save_interval* setting.

In case of a partition failure on the stable queues, be sure your setting allows adequate time to restore your system. Consider also the size of the partitions that are allocated for backlogged messages. Partitions must be large enough to hold the extra messages.

Refer to the *Replication Server Design Guide* capacity planning guidelines for help in determining queue space requirements.

Connections between Replication Servers and data servers

When you set the *save_interval* for a physical or logical connection between a Replication Server and a data server and database, you allow Replication Server to save transactions in the DSI queue. You can restore the backlogged transactions using the `sysadmin restore_dsi_saved_segments` command. Refer to the *Replication Server Reference Manual* for more information.

You can use these saved transactions to resynchronize a database after it has been loaded to a previous state from transaction dumps and database dumps.

For example, in Figure 19-1, if the replicate data server SYDNEY_DS that is connected to Replication Server SYDNEY_RS experiences a failure, it can obtain the messages saved in the DSI queue at SYDNEY_RS to resynchronize the replicate database after it has been restored.

You can also use the *save_interval* for setting up a warm standby of a database that holds some replicate data or one that receives applied functions.

Setting the save interval for connections

To set the *save_interval* for a database connection, execute the alter connection command at the Replication Server. For example, here is the command to set Replication Server SYDNEY_RS to save for one hour any messages destined for its replicate data server SYDNEY_DS.

```
alter connection to SYDNEY_DS.pubs2
set save_interval to '60'
```

By default, the *save_interval* is set to 0 (minutes).

You can also configure the save intervals for the DSI queue and the materialization queue for a logical connection. See “Configuring logical connection save intervals” on page 530 for details.

Backing up the RSSDs

If you cannot recover an RSSD’s most recent state, RSSD recovery can be complex. The procedure you use depends on how much RSSD activity there has been since the last dump. See Table 19-3 on page 649 for a list of possible recovery procedures.

You should perform a dump of your RSSDs following any replication DDL, such as changing routes or adding subscriptions.

Creating coordinated dumps

When you must recover a primary database by restoring backups, you must also make sure that replicate data in the affected databases at other sites is consistent with the primary data. To provide for consistency after a restore on multiple data servers, Replication Server provides a method for coordinating database dumps and transaction dumps at all sites in a replication system.

You initiate a database dump or transaction dump from the primary database. RepAgent retrieves the dump record from the log and submits it to Replication Server so that the dump request can be distributed to the replicate sites. The method ensures that all of the data can be restored to a known point of consistency.

You can only use a coordinated dump with databases that store either primary data or replicated data but not both. You initiate a coordinated dump from within a primary database.

The process for coordinating dumps works as follows:

- In each function-string class assigned to the databases involved, the Replication System Administrator at each site creates function strings for the `rs_dumpdb` and `rs_dumptran` system functions. The function strings should call stored procedures that execute the dump database and dump transaction or equivalent commands and update the `rs_lastcommit` system table. Refer to the *Replication Server Reference Manual* for examples.
- You must be using a function-string class, such as a derived class, in which you can create and modify function strings. See “Managing function-string classes” on page 450 for more information.
- Using the `alter connection` command, the Replication System Administrator at each replicate site configures the Replication Servers to enable a coordinated dump.
- When a dump is started in a primary database, the RepAgent transfers the dump database or dump transaction log record to the Replication Server.
- Replication Server distributes an `rs_dumpdb` or `rs_dumptran` function call to sites that have subscriptions for the replicated tables in the database.
- The `rs_dumpdb` and `rs_dumptran` function strings at the replicate sites execute the customized stored procedures at each replicate site.

Recovering from partition loss or failure

When a Replication Server detects a failed or missing partition, it shuts down the stable queues that are using the partition and logs messages about the failure. Restarting Replication Server does not correct the problem. You must drop the damaged partition and rebuild the stable queues.

Complete recovery depends on the volume of messages cleared from the queue and on how soon you apply the recovery procedure after the failure occurs. If a Replication Server maintains minimal latency in the replication system, only the most recent messages are lost when its queues are rebuilt.

If a partition fails in a primary Replication Server, you can usually resend lost messages from their source using an off-line database log. If partitions fail in a replicate Replication Server, you need to recover from the stable queue of the upstream Replication Server.

In some cases, using an off-line log may be the only way you can recover your messages. If the Replication Server has suspended routes or connections, or if a network or data server connection goes down, a backlog may have accumulated in the Replication Server’s stable queues. Unless you have specified a save interval setting that can cover the backlog, your chance of recovering these messages decreases with time. Source Replication Servers may have already deleted messages from their stable queues and may have truncated the database logs.

Note For details on setting and displaying the save interval for recovery purposes, see “Recovering from partition loss or failure” on page 637.

Table 19-2 summarizes when to use and where to locate the appropriate recovery procedure for partition loss or failure.

Table 19-2: Overview of symptoms and procedures

Symptom	Use this procedure
Replication Server detects lost, damaged, or failed stable queue.	“Procedure for recovering from partition loss or failure” on page 638.
Message loss occurred because a backlog existed in the failed Replication Server and there were insufficient messages saved at the previous site.	“Message recovery from off-line database logs” on page 640.
In addition to message loss, database logs have been truncated. Either the secondary truncation point is invalid or the <code>dbcc settrunc('ltn', 'ignore')</code> command, was executed to truncate log records that have not been transferred by RepAgent to the Replication Server.	Use “Truncated message recovery from the database log” on page 643 to recover the database log. Then use “Message recovery from off-line database logs” on page 640 to rebuild the stable queues and recover lost messages.

Procedure for recovering from partition loss or failure

To recover from Replication Server partition loss or failure, perform the following steps:

- 1 Log in to the Replication Server and drop the failed partition:

```
drop partition logical_name
```

Replication Server does not immediately drop a partition that was in use. If the partition is undamaged, Replication Server drops it only after all of the messages it holds are delivered and deleted.

Refer to “drop partition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information.

- 2 If the failed partition was the only one available to the Replication Server, add another one to replace it:

```
add partition logical_name
on 'physical_name' with size size
[starting at vstart]
```

Refer to “add partition” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information.

- 3 Since the partition is damaged, you must rebuild the stable queues:

```
rebuild queues
```

See “Rebuilding queues online” on page 664 for a description of this process.

When all stable queues on the partition are removed, Replication Server drops the failed partition from the system and rebuilds the queues using the remaining partitions.

- 4 After rebuilding the queues, check the Replication Server logs for loss detection messages.

See “Loss detection after rebuilding stable queues” on page 666 for background and details.

- 5 If Replication Server detected message loss, you can:

- Perform “Message recovery from off-line database logs” on page 640, or
- Request that Replication Server ignore the loss by executing the `ignore loss` command for the database on the Replication Server where the loss was detected.

Note If you specify that Replication Server ignore message losses and you have rebuilt the queues of a Replication Server that is part a route, you must recreate subscriptions at the destination or use the `rs_subcmp` program with the `-r` flag to reconcile primary and replicate data.

Message recovery from off-line database logs

If the online log does not contain all the data needed to recover, you must load an older version of the primary database into a separate database and start RepAgent for the database. Although RepAgent is accessing a different database, it submits messages as if they were from the database whose messages you are recovering.

To recover messages from off-line logs after a partition failure:

1 Restart Replication Server in standalone mode, using the -M flag.

2 Log in to the Replication Server, and enter:

```
rebuild queues
```

See “Rebuilding queues online” on page 664 for a description of this process.

3 Inspect the Replication Server logs at each site for “Checking Loss” messages.

See “Determining which dumps to load” on page 673 for background and details on examining these messages.

4 Use the date and time in the error log messages to determine which dumps to load.

5 Enable RepAgent for a temporary recovery database, using the `sp_config_rep_agent` system procedure.

```
sp_config_rep_agent temp_dbname, 'enable', \  
'rs_name', 'rs_user_name', 'rs_password'
```

See “Setting Replication Server configuration parameters” on page 99 for information about configuring RepAgent.

6 Load the database dump and the first transaction log dump in to a temporary recovery database.

7 Start RepAgent in recovery mode for the temporary database:

```
sp_start_rep_agent temp_dbname, 'for_recovery', \  
'connect_dataserver', 'connect_database', \  
'rs_name', 'rs_user_name', 'rs_password'
```

where “`connect_dataserver`” and “`connect_database`” specify the original primary data server and database.

RepAgent transfers data in the transaction log of the temporary recovery database to the original primary database. When RepAgent completes scanning the transaction log, it shuts down.

- 8 Verify that RepAgent has replayed the transaction log of the temporary database. Use either of these methods:
 - Check the Adaptive Server log for a message similar to the following:

```
Recovery of transaction log is complete. Please
load the next transaction log dump and then start
up the Rep Agent Thread with sp_start_rep_agent,
with 'for_recovery' specified.
```

Then, perform the appropriate actions.

- From Adaptive Server, execute the `sp_help_rep_agent` system procedure for recovery:

```
sp_help_rep_agent dbname, 'recovery'
```

This procedure displays RepAgent's recovery status. If the recovery status is "not running" or "end of log," then recovery is complete. You can load the next transaction log dump. If the recovery status is "initial" or "scanning," either the log has not been replayed, or the replay is not complete.

- 9 If you have performed another recovery procedure since you performed the last database dump, you may need to change the database generation number after loading a transaction log dump. See "Determining database generation numbers" on page 674.
- 10 If there are more transaction log dumps to load, repeat the following three steps for each dump:
 - a Load the next transaction log dump. (Be sure to load the dumps in the correct order.)
 - b Restart RepAgent in recovery mode.
 - c Watch the Adaptive Server log for the completion message or use `sp_help_rep_agent`.
- 11 Check the Replication Server logs for loss detection messages.

No losses should be detected unless you failed to load the database to a state old enough to retrieve all of the messages.

See "Loss detection after rebuilding stable queues" on page 666 for background and details.

- 12 Restart the Replication Server in normal mode.
- 13 Restart RepAgent for the original primary data server and database in normal mode.

Message recovery from the online database log

To recover messages that are still in the online log at the primary database, perform the following steps:

- 1 Stop all client activity.
- 2 Restart RepAgent for the primary database in recovery mode.

This process causes RepAgent to scan the log from the beginning so that it retrieves all messages.

Recovering from truncated primary database logs

This section describes how to recover from failures caused by truncating a primary transaction log before Replication Server has received the messages.

This situation typically occurs if RepAgent, a Replication Server (managing a primary database), or a network between them is down for a long time and RepAgent or Replication Server is unable to read records from the transaction log. The secondary truncation point cannot be moved, which prevents Adaptive Server from truncating the log and causes the transaction log of the primary database to fill up. You must then disable RepAgent by executing `sp_config_rep_agent` with the `disable` option turned on.

When a failed component returns to service, messages are missing at the Replication Server. Depending on the status of the lost messages, use one of the following procedures:

- If messages are still in the online log at the primary database (which is unlikely), see “Message recovery from the online database log” on page 642.
- If messages have been truncated from the online database log, see “Truncated message recovery from the database log” on page 643.

Truncated message recovery from the database log

In this procedure, you must load a previous database dump and transaction log dumps into a temporary recovery database. Then connect a RepAgent to that database to transmit the truncated log to the Replication Server. After the missing log records are recovered, you can restart the system using the regular primary database.

Using a temporary recovery database permits transaction recovery from clients that continued to use the primary database after its log was truncated.

Note Use the temporary database exclusively for recovering messages. Any modification to the database prevents you from loading the next transaction log dump. Also limit the activity on the original primary database so that the recovery can be completed before the transaction log on the original primary database must be dumped and truncated again.

To replay off-line transaction logs, follow these steps:

- 1 Create a temporary database such that the sysusages tables are similar in both the original and the temporary databases. To do this, you must use the same sequence of create database and alter database commands when creating the temporary database as were used to create the original database.
- 2 Shut down Replication Server.
- 3 Restart Replication Server in standalone mode, using the -M flag.
- 4 Log in to the Replication Server and execute the set log recovery command for each primary database you are recovering.

See “Setting log recovery for databases” on page 671.

This command puts the Replication Server into loss detection mode for the databases. Replication Server logs a message similar to the following:

```
Checking Loss for DS1.PDB from DS1.PDB
date=Nov-01-1995 10:35am
qid=0x01234567890123456789
```

- 5 Execute the `allow connections` command to allow Replication Server to accept connections only from other Replication Servers and from RepAgents in recovery mode.

Note If you attempt to connect to this Replication Server by automatically restarting RepAgent in normal mode with scripts, the Replication Server rejects the connection. You must restart RepAgent in recovery mode while pointing to the correct off-line log. This step allows you to resend old transaction logs before current transactions are processed.

- 6 Load the database dump into the temporary primary database.
- 7 Load the first or next transaction log dump into the temporary primary database.
- 8 Start the RepAgent for the temporary database in recovery mode:

```
sp_start_rep_agent temp_dbname, 'recovery',  
'connect_dataserver', 'connect_database',  
'repserver_name', 'repserver_username',  
'repserver_password'
```

where `connect_dataserver` and `connect_database` specify the original primary data server and database.

RepAgent transfers data in the transaction log of the temporary recovery database to the original primary database. When RepAgent completes scanning the current transaction log, it shuts down.

- 9 Verify that RepAgent has replayed the transaction log of the temporary database.

- a Check the Adaptive Server log for the following message:

```
Recovery of transaction log is complete. Please  
load the next transaction log dump and then start  
up the Rep Agent Thread with sp_start_rep_agent,  
with 'recovery' specified.
```

and perform the appropriate actions, or

- b Execute `admin who_is_down`.

If the RepAgent reports “down,” load the next transaction log.

- 10 Repeat steps 7 through 9 until all transaction logs have been processed.
You are now ready to resume normal replication from the primary database.

- 11 Shut down Replication Server, which is still in standalone mode.
- 12 Execute the following commands:

```
rs_zerolrm data_server, database
dbcc settrunc('ltm', 'valid')
```

Note You may need to execute `rs_zerolrm` to clear the locator information.

- 13 Restart Replication Server in normal mode.
- 14 Restart RepAgent for both the primary database and RSSD using `sp_start_rep_agent`.
- 15 If you have performed another recovery procedure since you performed the last database dump, you may need to change the database generation number after loading a transaction log dump. See “Determining database generation numbers” on page 674.

Recovering from primary database failures

Most database failures are recovered without losing any committed transactions. No special Replication Server recovery procedure is needed if the database recovers on restart—Replication Server performs a handshake with the database, ensuring that no transactions are lost or duplicated in the replication system.

If a primary database fails and you are unable to recover all committed transactions, you must load the database to a previous state and follow a recovery procedure designed to restore consistency at the replicate sites.

Here are two possible scenarios for recovering from primary database failures:

- Recovering with coordinated dumps

If you have coordinated dumps of primary and replicate databases, you can use them to load all databases in the replication system to a consistent state.

See “Loading from coordinated dumps” on page 646 for details.

- Recovering with primary dumps only

If you do not have coordinated dumps, you can load the failed primary database and then verify the consistency of the replicate databases with the restored primary database.

See “Loading a primary database from dumps” on page 647 for details.

Loading from coordinated dumps

Use this procedure only if you have coordinated dumps of both primary and replicate databases. To load a primary database and all replicate databases to the same state, follow this procedure:

- 1 Perform steps 1 through 15 from “Loading a primary database from dumps” on page 647.
- 2 Suspend connections to the replicate databases that must be restored.
- 3 For each replicate database, log in to its managing Replication Server and execute the suspend connection command:

```
suspend connection to data_server.database
```

- 4 Load the replicate databases from the coordinated dumps that correspond to the restored primary database state.
- 5 For each replicate database, log in to its managing Replication Server and execute a `sysadmin set_dsi_generation` command to set the generation number for the database to the same generation number used in step 1:

```
sysadmin set_dsi_generation, 101,  
primary_data_server, primary_database,  
replicate_data_server, replicate_database
```

The parameters *primary_data_server* and *primary_database* specify the primary database for loading. The parameters *replicate_data_server* and *replicate_database* specify the replicate database for loading.

Setting the generation numbers in this manner prevents Replication Servers from applying to the replicate databases any old messages that may be in the queues.

- 6 For each replicate database, log in to its managing Replication Server and execute the resume connection command to restart the DSI for the database:

```
resume connection to data_server.database
```

- 7 Restart the primary Replication Server in normal mode.

- 8 Restart RepAgent for the primary database in normal mode.

Note If any subscriptions were materializing when the failure occurred, drop them and re-create them.

Loading a primary database from dumps

Use this procedure if you are loading only a primary database in a replication system. To load the database to a previous state and resolve any inconsistencies with replicate databases, follow this procedure:

- 1 Log in to the primary Replication Server and use the admin `get_generation` command to get the database generation number for the primary database:

```
admin get_generation, data_server, database
```

Write down the generation number so you have it for step 7.

- 2 Shut down the RepAgent for the primary database. To do this execute `sp_stop_rep_agent` system procedure.

```
sp_stop_rep_agent database
```

- 3 Suspend the DSI connection to the primary database (for exclusive use).
- 4 Load the database to the most recent or previous state.

This step entails loading the most recent database dump and all subsequent transaction log dumps.

Refer to the *Adaptive Server Enterprise System Administration Guide* for instructions.

- 5 Resume the DSI connection.
- 6 Enter the following commands to dump the transaction log:

```
use database
go
dbcc settrunc('ltm', 'ignore')
go
dump tran database with truncate_only
go
dbcc settrunc('ltm', 'valid')
go
```

- 7 Execute the `dbcc settrunc` command in the restored primary database to set the generation number to the next higher number. For example, if the `admin get_generation` command in step 1 returned 0, enter the following commands:

```
use database
go
dbcc settrunc('ltm', 'gen_id', 1)
```

- 8 Run the following command to clear the locator information:

```
rs_zeroltm data_server, database
```

- 9 Start RepAgent for the primary database. To do this, execute the following command:

```
sp_start_rep_agent database
```

- 10 Run the `rs_subcmp` program for each subscription at the replicate sites. Use the `-r` flag to reconcile the replicate data with the restored primary data, or drop all the subscriptions and re-create them.

See Chapter 11, “Managing Subscriptions” in the *Replication Server Administration Guide Volume 1* for more information on using `rs_subcmp`. Also refer to Chapter 7, “Executable Programs,” in the *Replication Server Reference Manual* for more information about `rs_subcmp` command.

Recovering from RSSD failure

If you cannot recover the RSSD’s most recent database state, recovering from an RSSD failure is a complex process. In this case, you must load the RSSD from old database dumps and transaction log dumps.

The procedure for recovering an RSSD is similar to that for recovering a primary database. However, it requires more steps, since the RSSD holds information about the replication system itself. RSSD system tables are closely associated with the state of the stable queues and of other RSSDs in the replication system.

If a Replication Server’s RSSD has failed, you first need to determine the extent of recovery required. To do this, perform one or more of the following actions:

- When the RSSD becomes available, log in to the Replication Server and execute `admin who_is_down`. Some Replication Server threads may have shut down during the RSSD's period of inactivity.
 - If an SQM thread for an inbound or outbound queue or an RSI outbound queue is down, restart the Replication Server.
 - If a DSI thread is down, resume the connection to the associated database.
 - If an RSI thread is down, resume the route to the destination database.

RSM – For instructions on viewing thread status in Sybase Central, see “Viewing thread status” in Replication Server’s plug-in help.

- Check all connecting RepAgents to see if they are running with the `sp_help_rep_agent` system procedure. (RepAgents may have shut down in response to errors resulting from RSSD shutdown.) Restart them if necessary.
- If you cannot recover the RSSD’s most recent database state, you must load it from old database dumps and transaction log dumps. See “Recovering an RSSD from dumps” on page 649.

Recovering an RSSD from dumps

The procedure you use to recover an RSSD depends on how much RSSD activity there has been since the last RSSD dump. There are four increasingly severe levels of RSSD failure, with corresponding recovery requirements. Use Table 19-3 to locate the RSSD recovery procedure you need.

Table 19-3: Recovering from RSSD failures

Activity since last RSSD dump	Use this procedure
No DDL activity	“Basic RSSD recovery procedure” on page 650
DDL activity, but no new routes or subscriptions created	“Subscription comparison procedure” on page 652
DDL activity, but no new routes created	“Subscription re-creation procedure” on page 659
New routes created	“Deintegration/reintegration procedure” on page 662

Basic RSSD recovery procedure

Use the basic RSSD recovery procedure to restore the RSSD if you have executed no DDL commands since the last RSSD dump. DDL commands in RCL include those for creating, altering, or deleting routes, replication definitions, subscriptions, function strings, functions, function-string classes, or error classes.

Certain steps in this procedure are also referenced by other RSSD recovery procedures in this chapter.

Warning! Do not execute any DDL commands until you have completed this recovery procedure.

To perform basic RSSD recovery, follow these steps:

- 1 Shut down all RepAgents that connect to the current Replication Server.
- 2 Since its RSSD has failed, the current Replication Server is down. If for some reason it is not down, log in to it and use the shutdown command to shut it down.

Note Some messages may still be in the Replication Server stable queues. Data in those queues may be lost when you rebuild these queues in later steps.

- 3 Restore the RSSD by loading the most recent RSSD database dump and all transaction dumps.
- 4 Restart the Replication Server in standalone mode, using the `-M` flag.
You must start the Replication Server in standalone mode, because the stable queues are now inconsistent with the RSSD state. When the Replication Server starts in standalone mode, reading of the stable queues is not automatically activated.
- 5 Log in to the Replication Server, and get the generation number for the RSSD, using the `admin get_generation` command:

```
admin get_generation, data_server, rssid_name
```

For example, the Replication Server may return a generation number of 100.

- 6 In the Replication Server, rebuild the queues with the following command:

```
rebuild queues
```

See “Rebuilding queues online” on page 664 for a description of this process.

- 7 Start all RepAgents (except the RSSD RepAgent) that connect to the current Replication Server in recovery mode.

Wait until each RepAgent logs a message in the Adaptive Server log that it is finished with the current log.

- 8 Check the loss messages in the Replication Server log, and in the logs of all the Replication Servers with direct routes *from* the current Replication Server.
 - If all your routes were active at the time of failure, you probably will not experience any real data loss.
 - However, loss detection may indicate real loss. Real data loss may be detected if the database logs were truncated at the primary databases, so that the rebuild process did not have enough information to recover. If you have real data loss, reload database logs from old dumps. See “Recovering from truncated primary database logs” on page 642.
 - See “Loss detection after rebuilding stable queues” on page 666 for background and details on loss detection.
- 9 Shut down RepAgents for all primary databases managed by the current Replication Server.
- 10 Execute the `dbcc settrunc` command at the Adaptive Server for the restored RSSD. Move up the secondary truncation point.

```
use rssid_name
go
dbcc settrunc('ltm', 'ignore'
dump tran rssid_name with truncate_only
go
begin tran commit tran
go 40
```

Note The `begin tran commit tran go 40` command moves the Adaptive Server log onto the next page.

After completing step 10 and before continuing with step 11, run the following command to clear the locator information.

```
rs_zeroltm rssid_server, rssid_name
go
```

- 11 Execute the `dbcc settrunc` command at the Adaptive Server for the restored RSSD to set the generation number to one higher than the number returned by `admin get_generation` in step 5.

```
dbcc settrunc('ltm', 'valid')
go
```

Make a record of this generation number and of the current time, so that you can return to this RSSD recovery procedure, if necessary. Or, you can dump the database after setting the generation number.

- 12 Restart the Replication Server in normal mode.

If you performed this procedure as part of the subscription comparison or subscription re-creation procedure, the upstream RSI outbound queue may contain transactions, bound for the RSSD of the current Replication Server, that have already been applied using `rs_subcmp`. If this is the case, after starting the Replication Server, the error log may contain warnings referring to duplicate inserts. You can safely ignore these warnings.

- 13 Restart RepAgents for the RSSD and for user databases in normal mode.

If you performed this procedure as part of the subscription comparison or subscription re-creation RSSD recovery procedure, you should expect to see messages regarding RSSD losses being detected in all Replication Servers that have routes from the current Replication Server.

Subscription comparison procedure

Follow this RSSD recovery procedure if you have executed some DDL commands since the last transaction dump but you have not created any new subscriptions or routes. DDL commands in RCL include those for creating, altering, or deleting routes, replication definitions, subscriptions, function strings, functions, function-string classes, or error classes.

Warning! Do not execute any DDL commands until you have completed this recovery procedure.

Following this procedure makes the failed RSSD consistent with upstream RSSDs or consistent with the most recent database and transaction dumps (if there is no upstream Replication Server). It then makes downstream RSSDs consistent with the recovered RSSD.

If DDL commands have been executed at the current Replication Server since the last transaction dump, you may have to re-execute them.

Warning! This procedure may fail if you are operating in a mixed-version environment; that is, the Replication Servers in your replication system are not all at the same version level.

To restore an RSSD with subscription comparison, follow these steps:

- 1 To prepare the failed RSSD for recovery, perform steps 1 through 4 of “Basic RSSD recovery procedure” on page 650.
- 2 To prepare all upstream RSSDs for recovery, execute the `admin quiesce_force_rsi` command at each upstream Replication Server.
 - This step ensures that all committed transactions from the current Replication Server have been applied before you execute the `rs_subcmp` program.
 - Execute this command sequentially, starting with the Replication Server that is furthest upstream from the current Replication Server.
 - Make sure that RSSD changes have been applied, that is, that the RSSD DSI outbound queues are empty.
 - The Replication Server that is directly upstream from the current Replication Server cannot be quiesced.
- 3 To prepare all downstream RSSDs for recovery, execute the `admin quiesce_force_rsi` command at each downstream Replication Server.
 - This step ensures that all committed transactions bound for the current Replication Server have been applied before you execute the `rs_subcmp` program.
 - Execute this command sequentially, starting with Replication Servers that are immediately downstream from the current Replication Server.
 - Make sure that RSSD changes have been applied, that is, that the RSSD DSI outbound queues are empty.
- 4 Reconcile the failed RSSD with all upstream RSSDs, using the `rs_subcmp` program.
 - First execute `rs_subcmp` without reconciliation to get an idea of what operations it will perform. When you are ready to reconcile, use the `-r` flag to reconcile the replicate data with the primary data.

- You must execute `rs_subcmp` as the maintenance user. See Chapter 8, “Managing Replication Server Security” for more information on the maintenance user.
- In each instance, specify the failed RSSD as the replicate database.
- In each instance, specify the RSSD of each upstream Replication Server as the primary database.
- Start with the furthest upstream Replication Server, and proceed downstream for all other Replication Servers with routes (direct or indirect) to the current Replication Server.
- Reconcile each of the following RSSD system tables: `rs_articles`, `rs_classes`, `rs_columns`, `rs_databases`, `rs_erroractions`, `rs_functions`, `rs_funcstrings`, `rs_objects`, `rs_publications`, `rs_systext`, and `rs_whereclauses`.
- When you execute `rs_subcmp` on replicated RSSD tables, the `where` and `order by` clauses of the `select` statement must include all rows to be replicated. See “Using `rs_subcmp` on replicated RSSD system tables” on page 655 for more information.

The failed RSSD should now be recovered.

- 5 Reconcile all downstream RSSDs with the RSSD for the current Replication Server, which was recovered in the previous step, using the `rs_subcmp` program.
 - First execute `rs_subcmp` without reconciliation to get an idea of what operations it will perform. When you are ready to reconcile, use the `-r` flag to reconcile the replicate data with the primary data.
 - You must execute `rs_subcmp` as the maintenance user. See Chapter 8, “Managing Replication Server Security” for more information on the maintenance user.
 - In each instance, specify as the primary database the recovered RSSD.
 - In each instance, specify as the replicate database the RSSD of each downstream Replication Server.
 - Start with the Replication Servers that are immediately downstream, then proceed downstream for all other Replication Servers with routes (direct or indirect) from the current Replication Server.

- Reconcile each of the following RSSD system tables: `rs_articles`, `rs_classes`, `rs_columns`, `rs_databases`, `rs_erroractions`, `rs_functions`, `rs_funcstrings`, `rs_objects`, `rs_publications`, `rs_systext`, and `rs_whereclauses`.
- When you execute `rs_subcmp` on replicated RSSD tables, the `where` and `order by` clauses of the `select` statement must select all rows to be replicated. See “Using `rs_subcmp` on replicated RSSD system tables” on page 655 for more information.

All downstream RSSDs should now be fully recovered.

- 6 If the recovering Replication Server is an ID Server, you must restore the Replication Server and database IDs in its RSSD.
 - a For every Replication Server, check the `rs_databases` and `rs_sites` system tables for their IDs.
 - b Insert the appropriate rows in the recovering RSSD’s `rs_idnames` system table if they are missing.
 - c Delete from the recovering RSSD’s `rs_idnames` system table any IDs of databases or Replication Servers that are no longer part of the replication system.
 - d To ensure that the `rs_ids` system table is consistent, execute the following stored procedure in the RSSD of the current Replication Server:

```
rs_mk_rsids_consistent
```

- 7 If the recovering Replication Server is *not* an ID Server, *and* a database connection was created at the recovering Replication Server after the last transaction dump, delete the row corresponding to that database connection from the `rs_idnames` system table in the ID Server’s RSSD.
- 8 Perform steps 5 through 13 of “Basic RSSD recovery procedure” on page 650.
- 9 To complete RSSD recovery, re-execute any DDL commands executed at the current Replication Server since the last transaction dump.

Using `rs_subcmp` on replicated RSSD system tables

When executing `rs_subcmp` on replicated RSSD tables during RSSD recovery procedures, formulate the `where` and `order by` clauses of the `select` statement to select all rows that must be replicated for each system table.

Table 19-4 illustrates the general form of these select statements.

Note You may need to adjust these select statements in a mixed-version environment.

Table 19-4: select statements for rs_subcmp procedure

RSSD table name	select statement
rs_articles	select * from rs_articles where prsid in <i>sub_select</i> order by <i>primary_key</i>
rs_classes	select * from rs_classes where prsid in <i>sub_select</i> order by <i>primary_keys</i>
rs_columns	select * from rs_columns where prsid in <i>sub_select</i> and rowtype = 1 order by <i>primary_keys</i>
rs_databases	select * from rs_databases where prsid in <i>sub_select</i> and rowtype = 1 order by <i>primary_keys</i>
rs_erroractions	select * from rs_erroractions where prsid in <i>sub_select</i> order by <i>primary_keys</i>
rs_funcstrings	select * from rs_funcstrings where prsid in <i>sub_select</i> and rowtype = 1 order by <i>primary_keys</i>
rs_functions	select * from rs_functions where prsid in <i>sub_select</i> and rowtype = 1 order by <i>primary_keys</i>
rs_objects	select * from rs_objects where prsid in <i>sub_select</i> and rowtype = 1 order by <i>primary_keys</i>
rs_publications	select * from rs_publications where prsid in <i>sub_select</i> order by <i>primary_key</i>
rs_systext	select * from rs_systext where prsid in <i>sub_select</i> and texttype in ('O', 'S') order by <i>primary_keys</i>
rs_whereclauses	select * from rs_whereclauses where prsid in <i>sub_select</i> order by <i>primary_key</i>

In the select statements in Table 19-4, *sub_select* represents the following sub-selection statement, which selects all site IDs that are the source Replication Servers for the current Replication Server:

```
(select source_rsid from rs_routes
  where
    (through_rsid = PRS_site_ID
     or through_rsid = RRS_site_ID)
  and
    dest_rsid = RRS_site_ID)
```

where *PRS_site_ID* is the site ID of the Replication Server managing the primary RSSD, and *RRS_site_ID* is the site ID of the Replication Server managing the replicate RSSD for the rs_subcmp operation.

For the rs_columns, rs_databases, rs_funcstrings, rs_functions, and rs_objects system tables, if rowtype = 1, then the row is a replicated row. Only replicated rows need be compared using rs_subcmp.

For each system table, the *primary_keys* are its unique indexes.

Classes and system tables

The system-provided function-string classes and error class do not initially have a designated primary site, that is, their site ID equals 0. The classes `rs_default_function_class` and `rs_db2_function_class` cannot be modified, and thus can never have a designated primary site. The classes `rs_sqlserver_function_class` and `rs_sqlserver_error_class` may be assigned a primary site and modified. The primary site of a derived function-string class is the same as its parent class.

If the recovering Replication Server was made the primary site for a function-string class or error class since the last transaction dump, the `rs_subcmp` procedure described earlier in this section would find orphaned rows in downstream RSSDs.

In that event, run `rs_subcmp` again on the `rs_classes`, `rs_erroractions`, `rs_funcstrings`, and `rs_systext` system tables. Set `prsid = 0` in order to repopulate these tables with the necessary default settings. For example, use the following `select` statement for the `rs_classes` table:

```
select * from rs_classes where prsid = 0
       order by primary_keys
```

Example

Suppose you have the following Replication Server sites in your replication system, where an arrow (\rightarrow) indicates a route. Site B is the failed site, and there are no indirect routes.

- A > B
- C > B
- C > D
- B > E

These Replication Servers have the following site IDs:

- A = 1
- B = 2
- C = 3
- D = 4
- E = 5

In this example, to bring the RSSDs to a consistent state, you would perform the following tasks, in the order presented, on the `rs_classes`, `rs_columns`, `rs_databases`, `rs_erroractions`, `rs_funcstrings`, `rs_functions`, `rs_objects`, and `rs_systext` system tables.

Reconciling with upstream RSSDs

- 1 Run `rs_subcmp` against the above tables, specifying site B as the replicate and site A as the primary, with `prsid = 1` in the where clauses. For example, the select statement for `rs_columns` should look like the following:

```
select * from rs_columns where prsid in
(select source_rsid from rs_routes
where
    (through_rsid = 1 or through_rsid = 2)
    and dest_rsid = 2)
and rowtype = 1
order by objid, colname
```

- 2 Run `rs_subcmp` against the above tables, specifying site B as the replicate and site C as the primary, with `prsid = 3` in the where clauses. For example, the select statement for `rs_columns` should look like the following:

```
select * from rs_columns where prsid in
(select source_rsid from rs_routes
where
    (through_rsid = 3 or through_rsid = 2)
    and dest_rsid = 2)
and rowtype = 1
order by objid, colname
```

Reconciling downstream RSSDs

- 1 Run `rs_subcmp` against the above tables, specifying site B as the primary and site E as the replicate, with `prsid = 2` in the where clauses. For example, the select statement for `rs_columns` should look like the following:

```
select * from rs_columns where prsid in
(select source_rsid from rs_routes
where
    (through_rsid = 2 or through_rsid = 5)
    and dest_rsid = 5)
and rowtype = 1
order by objid, colname
```

Refer to “rs_subcmp” in Chapter 7, “Programs,” in the *Replication Server Reference Manual* for more information on rs_subcmp. Refer to Chapter 8, “Replication Server System Tables,” in the *Replication Server Reference Manual* for more information on the RSSD system tables.

Subscription re-creation procedure

Follow this RSSD recovery procedure if you have created new subscriptions or other DDL since the last transaction dump, and you have not created new routes. DDL commands in RCL include those for creating, altering, or deleting routes, replication definitions, subscriptions, function strings, functions, function-string classes, or error classes.

Warning! Do not execute any DDL commands until you have completed the subscription re-creation recovery procedure.

As with the subscription-comparison RSSD recovery procedure, following this procedure makes the failed RSSD consistent with upstream RSSDs or with the most recent database and transaction dumps (if there is no upstream Replication Server). It then makes downstream RSSDs consistent with the recovered RSSD.

In this procedure, however, you also either delete or re-create subscriptions that are in limbo due to the loss of the RSSD.

If DDL commands have been executed at the current Replication Server since the last transaction dump, you may have to reexecute them.

To restore an RSSD that requires that lost subscriptions be re-created, follow these steps:

- 1 To prepare the failed RSSD for recovery, perform steps 1 through 4 of “Basic RSSD recovery procedure” on page 650.
- 2 To prepare the RSSDs of all upstream and downstream Replication Servers for recovery, perform step 2 through 3 of “Subscription comparison procedure” on page 652.
- 3 Shut down all upstream and downstream Replication Servers affected by the previous step. Use the shutdown command.
- 4 Restart all upstream and downstream Replication Servers in standalone mode, using the -M flag.

All RepAgents connecting to these Replication Servers shut down automatically when you restart the Replication Servers in standalone mode.

- 5 To reconcile the failed RSSD with all upstream RSSDs, perform step 4 of “Subscription comparison procedure” on page 652.

The failed RSSD should now be recovered.

- 6 To reconcile all downstream RSSDs with the RSSD for the current Replication Server, perform step 5 of “Subscription comparison procedure” on page 652.
- 7 If the recovering Replication Server is an ID Server, to restore the IDs in its RSSD, perform step 6 of “Subscription comparison procedure” on page 652.
- 8 If the recovering Replication Server is *not* an ID Server *and* a database connection was created at the recovering Replication Server after the last transaction dump, perform step 7 of “Subscription comparison procedure” on page 652.
- 9 Query the `rs_subscriptions` system table of the current Replication Server for the names of subscriptions and replication definitions or publications and their associated databases.
 - Also query all Replication Servers with subscriptions to primary data managed by the current Replication Server, or with primary data to which the current Replication Server has subscriptions.
 - You can query the `rs_subscriptions` system table by using the `rs_helpsub` stored procedure.
- 10 For each user subscription in the `rs_subscriptions` system table, execute the `check subscription` command using the information obtained in step 9.
 - Execute this command at the current Replication Server and at all Replication Servers with subscriptions to primary data managed by the current Replication Server, or with primary data to which the current Replication Server has subscriptions.
 - Subscriptions with a status other than `VALID` must be deleted or re-created, as described below.
- 11 For each Replication Server that has a non-`VALID` subscription with the current Replication Server as the primary:
 - Note its `subid`, and delete the appropriate row from the primary `rs_subscriptions` system table.

- Use the subid from rs_subscriptions to find corresponding rows in the rs_rules system table, and also delete those rows.

For each system table, rs_subscriptions and rs_rules:

- If a subscription is in the primary table and not in the replicate table (because it was dropped), delete the subscription row from the primary table.
- If a subscription is in the replicate table and not in the primary table, delete the subscription row from the replicate table. After completing the rest of this procedure, re-create the subscription, as described in steps 17 through 19.
- If a subscription is in both the primary and replicate tables but is not VALID at one of the sites, delete the rows from both tables. After completing the rest of this procedure, re-create the subscription, as described in steps 17 through 19.

- 12 For each primary Replication Server for which the current Replication Server has a non-VALID user subscription:

- Note its subid, and delete the appropriate row from the primary rs_subscriptions system table.
- Use the subid from rs_subscriptions to find corresponding rows in the rs_rules system table, and also delete those rows.

For each system table, rs_subscriptions and rs_rules:

- If a subscription is in the primary table and not in the replicate table, delete the subscription row from the primary table. After completing the rest of this procedure, re-create the subscription, as described in steps 17 through 19.
- If a subscription is in the replicate table and not in the primary table (because it was dropped), delete the subscription row from the replicate table.
- If a subscription is in both the primary and replicate tables, but not VALID at one of the sites, delete the rows from both tables. After completing the rest of this procedure, re-create the subscription, as described in steps 17 through 19.

- 13 At both the primary and replicate Replication Server, execute the sysadmin drop_queue command for all existing materialization queues for subscriptions deleted in steps 17 through 19.

- 14 Restart in normal mode all Replication Servers, and their RepAgents, that had subscriptions to primary data managed by the current Replication Server or with primary data to which the current Replication Server had subscriptions.
- 15 Perform steps 5 through 13 of “Basic RSSD recovery procedure” on page 650.
- 16 Reexecute any DDL commands that executed at the current Replication Server since the last transaction dump.
- 17 Enable autocorrection for each replication definition.
- 18 Re-create the missing subscriptions using either the bulk materialization method or no materialization.

Use the define subscription, activate subscription, validate subscription, and check subscription commands for bulk materialization.
- 19 For each re-created subscription, restore consistency between the primary and replicate data in either of two ways:
 - Drop a subscription using the drop subscription command and the with purge option. Then re-create the subscription.
 - Use the rs_subcmp program with the -r flag to reconcile replicate and primary subscription data.

Refer to “rs_subcmp” in Chapter 7, “Programs,” in the *Replication Server Reference Manual* for more information on the rs_subcmp program. Refer to Chapter 8, “Replication Server System Tables,” in the *Replication Server Reference Manual* for more information on the RSSD system tables.

Deintegration/reintegration procedure

If you created routes since the last time the RSSD was dumped, you are required to perform the following tasks:

- 1 Remove the current Replication Server from the replication system.

See “Removing a Replication Server” on page 112 for details.
- 2 Reinstall the Replication Server.

Refer to the Replication Server installation and configuration guides for your platform for complete information on re-installing Replication Server.

- 3 Re-create Replication Server routes and subscriptions.

See Chapter 6, “Managing Routes” and Chapter 11, “Managing Subscriptions” for details.

Recovery support tasks

This section describes standard recovery tasks that are required in performing the recovery procedures described in this chapter. Use recovery tasks only for the procedure to which they apply. These tasks support recovery by letting you manipulate and identify critical data in the replication system.

Refer to this section for background in performing the recovery procedures in this chapter.

Table 19-5 lists the recovery support tasks.

Table 19-5: Overview of recovery support tasks

Recovery support task	See
Rebuild stable queues	“Rebuilding stable queues” on page 663
Check for Replication Server loss detection messages after rebuilding stable queues	“Loss detection after rebuilding stable queues” on page 666
Put Replication Server in log recovery mode	“Setting log recovery for databases” on page 671
Check for Replication Server loss detection messages after setting log recovery for databases	“Loss detection after setting log recovery” on page 672
Determine which dumps and logs to load	“Determining which dumps to load” on page 673
Adjust database generation numbers	“Adjusting database generation numbers” on page 674

Rebuilding stable queues

The rebuild queues command removes all existing queues and rebuilds them. It cannot rebuild individual stable queues.

You can rebuild queues online or off-line, depending on your situation. Generally, you rebuild queues online first to detect if there are lost stable queue messages. If there are lost messages, you can retrieve them by first putting the Replication Server in standalone mode and recovering the data from an off-line log.

Both methods for rebuilding queues are described in more detail in the following sections. Refer to “rebuild queues” in Chapter 3, “Replication Server Commands,” in the *Replication Server Reference Manual* for more information.

Rebuilding queues online

During the online rebuild process, the Replication Server is in normal mode. All RepAgents and other Replication Servers are automatically disconnected from the Replication Server. Connection attempts are rejected with the following message:

```
Replication Server is Rebuilding
```

Replication Servers and RepAgents retry connections periodically until rebuild queues has completed. At this time, the connections are successful.

When the queues are cleared, the rebuild is complete. The Replication Server then attempts to retrieve the cleared messages from the following sources:

- Other Replication Servers that have direct routes to the rebuilt Replication Server. If you have set a save interval from other Replication Servers, you have a greater likelihood of recovery.
- Database transaction logs for primary databases the Replication Server manages.

If there are loss detection messages, you need to check the status of these messages. Depending on the failure condition, if these messages are no longer available at their source, you may need to rebuild the queues using off-line logs. Or, you can request that Replication Server ignore the lost messages. See “Rebuilding queues from off-line database logs” on page 664 and “Loss detection after rebuilding stable queues” on page 666.

Rebuilding queues from off-line database logs

This task is used to recover data from off-line database logs. You use the rebuild queues command only after you have restarted the Replication Server in standalone mode. For details on standalone mode, see “Using standalone mode” on page 665. Executing rebuild queues in standalone mode puts Replication Server in recovery mode.

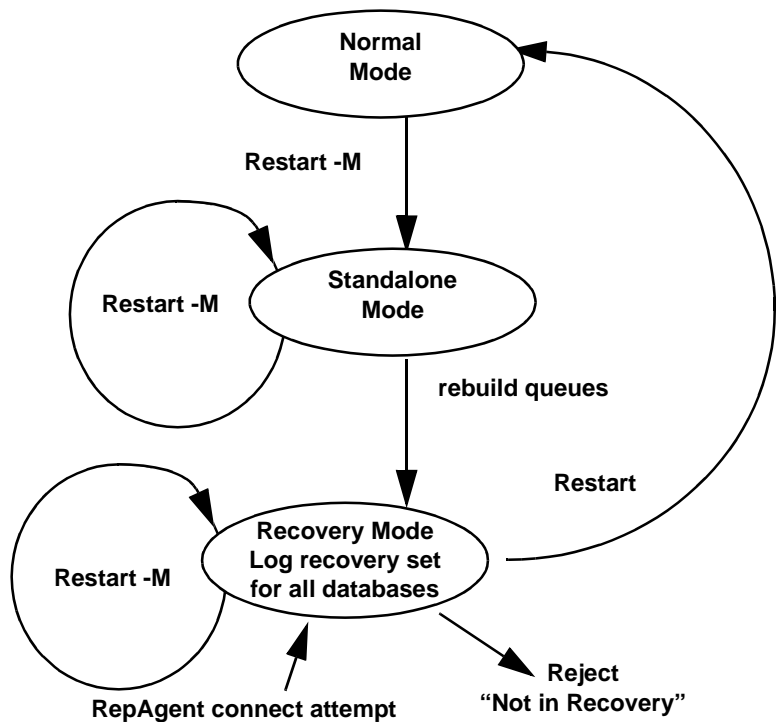
In recovery mode, the Replication Server allows only RepAgents in recovery mode to connect. If a RepAgent that is not in recovery mode attempts to connect, Replication Server rejects it with following error message:

Rep Agent not in recovery mode

If you use a script that automatically restarts RepAgent and connects it to the Replication Server, you must start RepAgent using the `for_recovery` option. RepAgents are not allowed to connect in normal mode.

Figure 19-2 illustrates the progression from normal mode to standalone mode to recovery mode using the `rebuild queues` command.

Figure 19-2: Entering recovery mode with the `rebuild queues` command



Using standalone mode

To start Replication Server in standalone mode, use the `-M` flag. Standalone mode is useful for looking at the state of Replication Server because the state is static. Standalone mode allows you to review the contents of the stable queues because no messages are being written to or read from the queues.

Standalone mode differs from Replication Server's normal mode in the following ways:

- No incoming connections are accepted. If any RepAgent or Replication Server attempts to connect to a Replication Server in standalone mode, the message “Replication Server is in Standalone Mode” is raised.
- No outgoing connections are started. A Replication Server in standalone mode does not attempt to connect to other Replication Servers.
- No DSI threads are started, even if there are messages in the DSI queues that have not been applied.
- No Distributor (DIST) threads are started. A DIST thread reads messages from the inbound queues, performs subscription resolution, and writes messages to the outbound queues.

Loss detection after rebuilding stable queues

To determine if any messages could not be recovered after the stable queues were rebuilt, the Replication Server performs loss detection. By checking Replication Server loss-detection messages, you can determine what kind of user intervention, if any, is necessary to restore all data to the system.

Replication Server detects two types of losses after rebuilding stable queues:

- SQM loss, which refers to data lost between two Replication Servers, detected at the next *downstream* site
- DSI loss, which refers to data lost between a Replication Server and a replicate database that the Replication Server manages

Both kinds of loss detection are addressed in the following sections.

If all data is available, no intervention is necessary and the replication system can return to normal operations. For example, if you know that the save interval for the route or connection is set for a longer length of time than the failure, you can recover all messages with no intervention. However, if the save interval is not set or is set too low, some messages may be lost.

Note A Replication Server that has detected a loss does not accept messages from the source. Loss detection prevents the source from truncating its stable queues. For example, if Replication Server RS2 detects that replicate data server DS2.RDB has lost data from primary data server DS1.PDB, Replication Server RS1 cannot truncate its queues until you decide how to handle the loss.

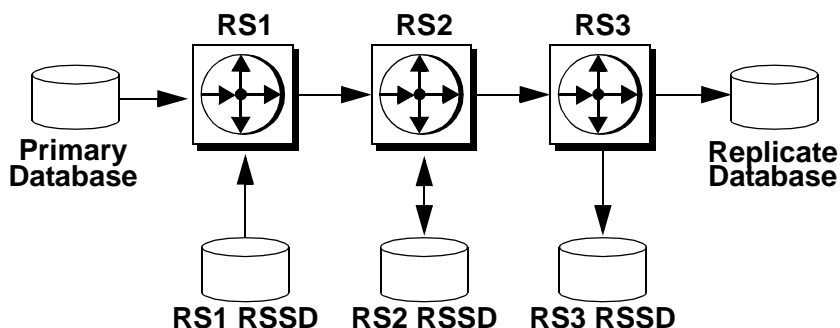
As a result, RS1 may run out of stable storage. Before a loss is detected (that is, after the “Checking Loss” message is reported), you can choose to ignore losses for a source and destination pair.

SQM loss between two Replication Servers

Every time you rebuild stable queues during a recovery procedure, Replication Server requests backlogged messages from sites that send its distributions. If the Replication Server manages primary databases, it instructs their RepAgents to send messages from the beginning of the online transaction logs. The backlogged messages repopulate the emptied stable queues.

Replication Server enables loss detection mode at those sites you are rebuilding that have a direct route from the Replication Server. In Figure 19-3, Replication Server RS3 detects losses if you rebuild the queues of Replication Server RS2. Similarly, RS2 detects losses if you rebuild the queues of Replication Server RS1.

Figure 19-3: Replication system loss detection example



When you execute the rebuild queues command at RS2, RS3 performs loss detection for all primary databases whose updates are routed to RS3 through RS2. RS3 logs messages for each of these databases. If you rebuild queues at RS3, no SQM loss detection is performed, because there are no routes originating from RS3.

Replication Server detects loss by looking for duplicate messages. If RS3 receives a message that it had received before the rebuild queues command, then no messages were lost. If the first message RS3 receives after rebuild queues has not been seen before, then either messages were lost, or no messages were in the stable queue.

Even if there are no messages in the stable queue from a specific source, RS3 identifies them as lost because it has no duplicate messages to use for a comparison. You can prevent this false loss detection by creating a heartbeat with an interval that is less than the save interval. This guarantees that there will always be at least one message in the stable queue.

RSM

For instructions on using heartbeats in Sybase Central, see “Concepts for using heartbeats to monitor latency” in Replication Server’s plug-in help.

SQM example

When RS3 performs SQM loss detection for the rebuilt RS2, it logs in its log file messages similar to the following “Checking Loss” message examples. These messages mark the beginning of the loss detection process. Subsequent messages are logged with the results. Each message contains a source and destination pair.

The first example message indicates that RS3 is checking loss for the RSSD at RS3 from the RSSD at RS2:

```

Checking Loss for DS3.RS3_RSSD from DS2.RS2_RSSD
date=Nov-01-95 10:15 am
qid=0x01234567890123456789
  
```

The second example message indicates that RS3 is checking loss for the replicate database RDB at RS3, from the primary database PDB at RS1:

```
Checking Loss for DS3.RDB from DS1.PDB
date=Nov-01-95 11:00am
qid=0x01234567890123456789
```

The third example message indicates that RS3 is checking loss for the RSSD at RS3 from the RSSD at RS1:

```
Checking Loss for DS3.RS3_RSSD from DS1.RS1_RSSD
date=Nov-01-95 10:00am
qid=0x01234567890123456789
```

RS3 reports whether it detects a loss. For example, the results of such loss-detection tests might read as follows:

```
No Loss for DS3.RS3_RSSD from DS2.RS2_RSSD
Loss Detected for DS3.RDB from DS1.PDB
No Loss for DS3.RS3_RSSD from DS1.RS1_RSSD
```

DSI loss between a Replication Server and its databases

Some messages in Replication Server queues are destined for databases, rather than for other Replication Servers. The DSI performs loss detection in a way that is similar to stable queue loss detection.

If you rebuild queues at a Replication Server that has no originating routes, no SQM loss detection is performed, but the Replication Server performs DSI loss detection for its messages.

DSI example

The DSI at Replication Server RS2 generates the following message for the RSSD at RS2:

```
DSI: detecting loss for database DS2.RS2_RSSD from
origin DS1.RS1_RSSD
date=Nov-01-95 10:58pm
qid=0x01234567890123456789
```

When retained messages begin arriving from previous sites, the DSI detects a loss, depending on whether the first message from the origin has already been seen by the DSI. If it detects no loss, a message similar to the following one is generated:

```
DSI: no loss for database DS2.RS2_RSSD from origin
DS1.RS1_RSSD
```

If the DSI does detect a loss, a message like the following one is generated:

```
DSI: loss detected for database DS2.RS2_RSSD from origin
```

DS1.RS1_RSSD

Handling losses

When Replication Server detects a loss, no further messages are accepted on the connection to the SQM or the DSI.

For example, when RS3 detects an SQM message loss for the RDB database from the PDB database, it rejects all subsequent messages from the PDB database to the RDB database.

Recovering a loss

To recover the loss, you need to choose one of the following options:

- Ignore the loss and continue, even though some messages may be lost. You can use the `rs_subcmp` program with the `-r` flag to reconcile primary and replicate data.

To run `rs_subcmp`, see “Subscription comparison procedure” on page 652. See also Chapter 11, “Managing Subscriptions” and refer to “`rs_subcmp`” in Chapter 7, “Programs,” in the *Replication Server Reference Manual*.

- Ignore the loss, then drop and re-create the subscriptions.
- Recover by replaying transactions from off-line logs (primary Replication Server loss only). In this case, you are not ignoring the loss.

Ignoring a loss

You must execute an ignore loss command in the following situations:

- If you choose to recover the lost messages by re-creating subscriptions or replaying logs.
- For an SQM loss, at the Replication Server that reported that loss, to force the Replication Server to begin accepting messages again. For example, to ignore a loss RS3 detected from DS1.PDB, enter the following command at RS3:

```
ignore loss from DS1.PDB to DS3.RDB
```

- For a DSI loss, at the database on the Replication Server where the loss was detected. For example, to ignore a loss reported in DS2.RS2_RSSD from origin DS1.RS1_RSSD, enter the following command at RS2:

```
ignore loss from DS1.RS1_RSSD to DS2.RS2_RSSD
```

- For both an SQM and a DSI loss that is detected by a Replication Server at the destination of the route when you rebuild two Replication Servers in succession.

In this case, you need to execute `ignore loss` twice, once for SQM losses and once for DSI losses. The `ignore loss` command that you execute to ignore DSI loss at the destination Replication Server is the same command you use to ignore SQM loss from the previous site.

Setting log recovery for databases

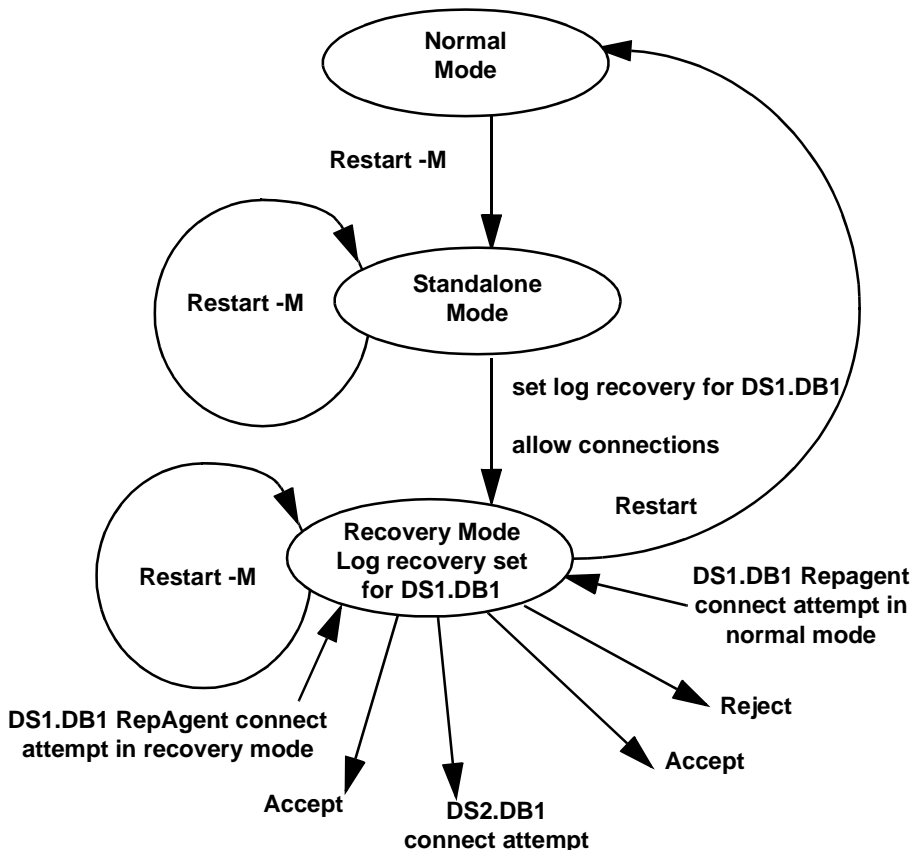
Setting log recovery manually is part of the procedure for recovering from truncated primary database logs off-line or restoring primary and replicate databases from dumps. While the procedure to rebuild queues off-line automatically sets log recovery for all databases, setting log recovery manually allows you to recover each database without reconstructing the stable queue.

The `set log recovery` command places Replication Server in log recovery mode for a database. You execute this command after placing Replication Server in standalone mode. To connect the RepAgents only to those databases that have been set for log recovery mode, execute the `allow connections` command. This puts the Replication Server in recovery mode.

Figure 19-4 illustrates the progression from normal mode to standalone mode to recovery mode using the `set log recovery` and `allow connections` commands.

For databases specified with the `set log recovery` command, Replication Server only accepts connections from other Replication Servers and from RepAgents that are in recovery mode. You then recover the transaction dumps into a temporary recovery database.

Figure 19-4: Entering recovery mode with the allow connections command



Loss detection after setting log recovery

While you are applying the temporary recovery database to the primary database, Replication Server may detect SQM loss between a primary database and the Replication Server that manages that primary database.

If all data is available, no intervention is necessary and the replication system can return to normal operations. The Replication Server logs a message such as:

```
No Loss Detected for DS1.PDB from DS1.PDB
```

If there were not enough messages, Replication Server logs a loss detection message similar such as:

```
Loss Detected for DS1.PDB from DS1.PDB
```

You must decide whether to ignore the losses by executing the ignore loss command, or repeat the recovery procedure from the beginning. To ignore the loss, enter the following command at the primary Replication Server:

```
ignore loss for DS1.PDB from DS1.PDB
```

If you received loss detection messages, you failed to reload the database to a state old enough to retrieve all of the messages. See “Determining which dumps to load” on page 673.

Determining which dumps to load

When loading transaction log dumps, always examine the “Checking Loss” message that is displayed during loss detection. If there is more than one message, choose the earliest date and time to determine which dumps to load.

For example, if the following message is generated by a Replication Server, you would load the dumps taken just before November 1, 1995 at 10:58 p.m.:

```
Checking Loss for DS3.RDB from DS1.PDB
date=Nov-01-1995 10:58pm
qid=0x01234567890123456789
```

The date in the message is the date and time of the oldest open transaction in the log when the last message received by the Replication Server was generated by the origin queue. Locate the most recent transaction dump with a timestamp before the date and time in the message. Then find the full database dump taken before that transaction dump.

The origin queue ID, or qid, is formed by the RepAgent and identifies a log record in the transaction log. The date is embedded in the qid as a timestamp. Replication Server converts the timestamp to a date for RepAgents for Adaptive Server.

Replication agents for non-Sybase data servers may also embed the timestamp in the qid. Replication Server converts the timestamp for non-Sybase data servers in bytes 20–27. The use of these bytes depends on the replication agent.

Note If the data server is not an Adaptive Server, the date in the message may appear nonsensical. You may need to decode the qid in bytes 20–27 to identify the dumps to load.

Adjusting database generation numbers

Each primary database in a replication system includes a database generation number. This number is stored both in the database and in the RSSD of the Replication Server that manages the database.

Any time you load a database for recovery, you may be required to change the database generation number, as instructed in the recovery procedure you are using. This section explains this step.

Determining database generation numbers

RepAgent for a primary database places the database generation number in the high-order 2 bytes of the qid that it constructs for each log record it passes to the Replication Server.

The remainder of the qid is constructed from other information that gives the location of the record in the log and also ensures that the qid increases for each record passed to Replication Server.

The requirement for increasing qid values allows Replication Server to detect duplicate records. For example, when a RepAgent restarts, it may resend some log records that Replication Server has already processed. If Replication Server receives a record with a lower qid than the last record it processed, it treats the record as a duplicate and ignores it.

If you are restoring a primary database to an earlier state, increment the database generation number so that the Replication Server does not ignore log records submitted after the database is reloaded. This step applies only if you are using the procedures described in “Loading a primary database from dumps” on page 647 or in “Loading from coordinated dumps” on page 646.

If you are replaying log records, increment the database generation number only if RepAgent previously sent the reloaded log records with the higher generation number. This situation arises only if you have to restore the database and log to a previous state for the first failure and then later replay the log due to a second failure.

Warning! Only change the database generation number as part of a recovery procedure. Changing the number at any other time can result in duplicate or missing data at replicate databases.

Dumps and database generation numbers

When you reload a database dump, the database generation number is included in the restored database. Since the database generation number is also stored in the RSSD of the Replication Server that manages the database, you may need to update that number so that it matches the one in the restored database.

However, when you reload a transaction log, the database generation number is not included in the restored log. For example, assume the following operations have occurred in a database:

Table 19-6: Dumps and database generation numbers

Operation	Database generation number
database dump D1	100
transaction dump T1	100
dbcc settrunc('ltm', 'gen_id', 101)	101
transaction dump T2	101
database dump D2	101

If you reload database dump D1, database generation number 100 is restored with it. If you reload transaction dump T1, the generation number remains at 100. After transaction dump T2, the generation number remains at 100, because reloading transaction dumps does not alter the database generation number. In this case, you need to change the database generation number to 101 using the `dbcc settrunc` command before having RepAgent scan transaction dump T2.

However, if you load database dump D2 before resuming replication, you do not have to alter the database generation number, since the number 101 is restored.

Asynchronous Procedures

This appendix describes asynchronous stored procedures.

Name	Page
Overview	677
Applied stored procedures	679
Request stored procedures	680
Asynchronous stored procedure prerequisites	681
Steps for implementing an applied stored procedure	682
Steps for implementing a request stored procedure	686
Specifying stored procedures and tables for replication	688
Managing user-defined functions	689

This appendix describes the method for replicating stored procedures that are associated with *table replication definitions*. This method is supported for applications that require it.

See Chapter 10, “Managing Replicated Functions” for information about replicated stored procedures that are associated with *function replication definitions*. The method described in that chapter is the recommended method for replicating stored procedures.

Refer to *Replication Server Design Guide* for more information on replication system design issues relating to replicated stored procedures.

Overview

Asynchronous procedure delivery allows you to execute SQL stored procedures that are designated for replication at primary or replicate databases. Because these stored procedures are marked for replication using the `sp_setreplicate` or `sp_setreproc` system procedures, they are called replicated stored procedures.

To satisfy the requirements of distributed applications, Replication Server provides two types of asynchronous stored procedure delivery: applied stored procedures and request stored procedures. Each type is described in this appendix.

Logging replicated stored procedures

Adaptive Server uses the following method to determine in which database a replicated stored procedure execution will be logged:

The procedure gets logged in the database in which the enclosing transaction was started.

- If the user does not begin a transaction explicitly, Adaptive Server will begin one in the user's current database before the stored procedure execution.
- If the user begins the transaction in one database, and then executes a replicated stored procedure in another database, the execution will still be logged in the database where the user began the transaction.

If the execution of a table-style replicated stored procedure (marked for replication by using either `sp_setreplicateproc_name, 'true'` or `sp_setrepprocproc_name, 'table'`) is logged in one database and changes replicated tables in another database, the table's changes and the procedure execution are logged in different databases. Therefore, the effects of the stored procedure execution can be replicated twice. The first time the stored procedure execution itself is replicated. The second time table changes that have been logged in the other database are replicated.

Logging replicated stored restrictions

Note that replicated Adaptive Server stored procedures may not contain parameters with the text and image datatypes. Refer to the *Adaptive Server Reference Manual* for more information.

Mixed-mode transactions

If a single transaction that invokes one or more request stored procedures is a mixed-mode transaction that also executes applied stored procedures or contains data modification language, Replication Server processes the request stored procedures after all the other operations. All request operations are processed together in a single separate transaction. This situation may arise where a single Replication Server manages both primary and replicate data.

Applied stored procedures

Replicated stored procedures that Replication Server delivers from a primary database to a replicate database are called applied stored procedures.

You use applied stored procedure delivery to replicate transactions first performed on primary data to replicate databases. Data changes are applied at a primary database and then distributed at a later time to replicate databases that subscribe to replication definitions for the data. Replication Server executes the replicated stored procedure in the replicate database as the maintenance user, which is consistent with normal data replication.

You can use applied stored procedures to realize important performance benefits. For example, if your organization has a large amount of row changes, you can create an applied stored procedure which changes many rows, rather than replicating the rows individually. You can also use applied stored procedures to replicate data set changes which are difficult to express using normal subscriptions. Refer to the *Replication Server Design Guide* for more information.

You set up applied stored procedures by making the first statement in the stored procedure update a table. You must also make sure that the destination databases have subscriptions to the before and after images of that updated row. The applied stored procedure must update only one row in a replicated table. Replication Server uses the first row updated by the stored procedure to determine where to send the user-defined function for the procedure.

If the rules in setting up the applied stored procedure are not met, Replication Server fails to distribute the stored procedure to replicate databases. See “Warning conditions” on page 684 for a list of actions that Replication Server takes if it fails to deliver the applied stored procedure.

Request stored procedures

Replicated stored procedures that Replication Server delivers from a replicate database to a primary database are called request stored procedures. You use a request stored procedure to deliver a transaction from a replicate database back to the primary database.

For example, a client application at a remote location may need to make changes to primary data. In this case, the application at the remote location executes a request stored procedure locally to change the primary data. Replication Server delivers this request stored procedure to the primary database by executing, in the replicate database, a stored procedure that has the same name as the stored procedure in the primary database. The stored procedure in the primary database updates the primary data that the transaction changes.

Replication Server executes the replicated stored procedure in the primary database as the user who executed the stored procedure in the replicate database. This ensures that only authorized users may change primary data.

In an application, Replication Server may replicate some or all of the data that is changed in the primary database. The changes are propagated to replicate databases managed by Replication Servers with subscriptions for the related data, either as data rows (insert, delete, or update operation) or as stored procedures. Using this mechanism, the effect of a transaction quickly arrives at both the primary and replicate databases.

Warning! Do not execute a request stored procedure in a primary database. This can lead to looping behavior, in which replicate Replication Servers cause the same procedure to execute in the primary database.

Using request stored procedures ensures that all updates are made at the primary database, preserving Replication Server's basic primary copy data model while keeping the replication system invulnerable to network failures and excess traffic. Even when there is primary database failure, or network failure from the replicate database to the primary database, Replication Server remains fault tolerant. It queues any undelivered request stored procedure invocations until the failed components come back online. When the components are again in service, Replication Server completes delivery.

By using Replication Server's guaranteed request stored procedure delivery feature, you can obtain all the benefits of having a single, definitive copy of your data that includes all the latest changes. At the same time, Replication Server provides the availability and performance benefits of de-coupling applications at replicate databases from the primary database.

Refer to the *Replication Server Design Guide* for more information on replication system design issues relating to asynchronous procedure delivery.

Asynchronous stored procedure prerequisites

Before implementing applied or request stored procedures on your system, be sure you:

- Understand how you will use asynchronous procedure delivery to meet your application needs. Refer to the *Replication Server Design Guide* for more information.
- Set up a RepAgent or LTM for the stored procedure, even if the database contains no primary data (such as when using request functions). Refer to the Replication Server installation and configuration guides for your platform for details.
- Create a function string for user-defined functions for function-string classes for which Replication Server does not generate default function strings. You can use the alter function string command to replace a default function string with one that performs the action your application requires.

See "Function strings and function-string classes" on page 457 for more information.

- Follow the step-by step instructions provided in this chapter for setting up applied or request stored procedures.

Note For function-string classes for which default generated function strings are provided, Replication Server creates a default function string that executes a stored procedure with the same name as the user-defined function. The procedures in this chapter assume that Replication Server processes applied or request stored procedures for such classes. For all other classes, you must create function strings for the user-defined function string.

Steps for implementing an applied stored procedure

To implement an applied stored procedure, perform the following steps:

- 1 Review the requirements described in “Asynchronous stored procedure prerequisites” on page 681.
- 2 Set up replicate databases that contain replicate tables. These tables may or may not match the replication definition for the primary table.
- 3 As necessary, set up routes from the primary Replication Server to the replicate Replication Servers that have subscriptions to replication definitions for the primary table.

See Chapter 6, “Managing Routes” for details on setting up routes.

- 4 Locate or create a replication definition on the primary Replication Server that identifies the table to be modified.

See Chapter 9, “Managing Replicated Tables” for information on creating replication definitions.

- 5 In the primary database, use the `sp_setreplicate` system procedure or the `sp_setreptable` system procedure to mark the table for replication. For example, for a table named `employee`:

```
sp_setreplicate employee, 'true'
```

or

```
sp_setreptable employee, 'true'
```

For `sp_setreptable`, the single quotes are optional.

See “Specifying stored procedures and tables for replication” on page 688 for details on using `sp_setreplicate`. See “Using the `sp_setreptable` system procedure” on page 276 for details on using `sp_setreptable`.

- 6 Create the stored procedure on the primary database. The first statement in the stored procedure must contain an update command for the first row of the primary table. For example:

```
create proc upd_emp
  @emp_id int, @salary float
as
update employee
set salary = salary * @salary
where emp_id = @emp_id
```

Warning! If the first statement in the stored procedure contains an operation other than update, Replication Server cannot distribute the stored procedure to replicate databases. See “Warning conditions” on page 684 for more information.

Never include dump transaction or dump database commands in the stored procedure. If the stored procedure contains commands with statement level errors, the error may occur at the replicate DSI. Depending on the error actions, the DSI may shut down.

- 7 In the primary database, use the `sp_setreplicate` system procedure or the `sp_setrepproc` system procedure to mark the stored procedure for replication. For example:

```
sp_setreplicate upd_emp, 'true'
```

or

```
sp_setrepproc upd_emp, 'table'
```

See “Specifying stored procedures and tables for replication” on page 688 for details on using `sp_setreplicate`. See “Marking stored procedures for replication” on page 343 for details on using `sp_setrepproc`.

- 8 At the replicate Replication Servers, create subscriptions to a replication definition for the table that the stored procedure at the primary database updates.

See Chapter 11, “Managing Subscriptions” for details on creating subscriptions.

Warning! Be sure the replicate database subscribes to both the before image and after image of the updated row. If it does not, Replication Server cannot distribute the stored procedure to the replicate database. See “Warning conditions” on page 684 for more information.

- 9 Create a stored procedure on the replicate database with the same name and parameters as the stored procedure on the primary database, but do not mark the procedure as replicated. For example:

```
create proc upd_emp
  @emp_id int, @salary float
as
update employee
set salary = salary * @salary
where emp_id = @emp_id
```

- 10 Grant execute permission on the stored procedure to the maintenance user. For example:

```
grant execute on upd_emp to maint_user
```

- 11 Create a user-defined function on the primary Replication Server that associates the stored procedure to the name of a replication definition for the table it updates. For example:

```
create function employee_rep.upd_emp  
(@emp_id int, @salary float)
```

Only one user-defined function are shared by all replication definitions for the same table. You can specify the name of any of these replication definitions.

- 12 Verify that all Replication Server and database objects in steps 1 through 11 exist at the appropriate locations.

Refer to Chapter 6, “Adaptive Server Stored Procedures,” in the *Replication Server Reference Manual* for information about stored procedures used to query the RSSD for system information.

Warning conditions

If the first statement in the applied stored procedure is an operation other than update, or the replicate database does not subscribe to the before image and after image of the updated row, Replication Server fails to deliver the applied stored procedure to the replicate database. Instead, Replication Server performs other actions that you can interpret as warnings.

The actions Replication Server takes is based on:

- The first operation (other than update) contained in the applied stored procedure at the primary database
- Whether the row modification stays in the subscription for the replicate database, and whether it matches the subscription’s before image or after image

Conditions and actions

This section identifies the warning conditions that prevent Replication Server from delivering an applied stored procedure at a replicate database.

Condition: The first row operation is an insert operation.

Action: Replication Server distributes the insert operation instead of the applied stored procedure.

Condition: The first row operation is a delete operation.

Action: Replication Server distributes the delete operation instead of the applied stored procedure.

Condition: Replicate Replication Servers have subscriptions that match the before image, but not the after image, of the modified row.

Action: Replication Server distributes a delete operation (`rs_delete` system function) to replicate databases with subscriptions to the before image but not the after image of the row modification.

Example: Assume there is a table T1 that has a column named C1 with a value of 1. A replicate database has a subscription to a replication definition for table T1 where $C1 = 1$.

If the associated stored procedure is executed with the parameters = 1 (before image) and = 2 (after image), the replicate database does not subscribe to the after image value of 2. Therefore, Replication Server distributes the delete operation to the replicate database.

Condition: Replicate Replication Servers have subscriptions that match the after image, but not the before image of the modified row.

Action: Replication Server distributes an insert operation (`rs_insert` system function) to replicate databases with subscriptions to the after image but not the before image of the row modification.

Example: Assume there is a table T1 that has a column named C1 with a value of 1. A replicate database has a subscription to a replication definition for table T1 where $C1 = 2$.

If the associated stored procedure is executed with the parameters = 1 (before image) and = 2 (after image), the replicate database does not subscribe to the before image value of 1. Therefore, Replication Server distributes the insert operation to the replicate database.

Condition: Replicate Replication Servers have subscriptions that match neither the before image nor the after image of the row modification.

Action: Replication Server does not distribute any operation or stored procedure to the replicate databases.

Example: Assume there is a table T1 that has a column named C1 with a value of 1. A replicate database has a subscription to a replication definition for table T1 where $C1 > 2$.

If the associated stored procedure is executed with the parameters equal to 1 (before image) and equal to 2 (after image), the replicate Replication Server does not subscribe to either the before image value of 1 or the after image value of 2. Therefore, Replication Server performs no distribution to the replicate database.

Steps for implementing a request stored procedure

To implement a request stored procedure, perform the following steps:

- 1 Review the requirements described in “Asynchronous stored procedure prerequisites” on page 681.
- 2 As necessary, set up a route from the replicate Replication Server to the primary Replication Server where the data is updated, and from the primary Replication Server to the replicate Replication Server that sends the update.

See Chapter 6, “Managing Routes” for details on setting up routes.

- 3 Create a login name and password at the primary Replication Server for the user at the replicate Replication Server.

See Chapter 8, “Managing Replication Server Security” for details.

- 4 At the replicate Replication Server, create the necessary permissions for this user to execute the stored procedure at the primary Replication Server.

See Chapter 8, “Managing Replication Server Security” for details.

- 5 At the primary Replication Server, locate or create a replication definition that identifies the table to be modified.

See Chapter 9, “Managing Replicated Tables” for information on creating replication definitions.

The replicate Replication Server may have subscriptions on the replication definition.

- 6 Create the stored procedure, which does not perform any updates, on the replicate database. For example:

```
create proc upd_emp
  @emp_id int, @salary float
as
print "Transaction accepted."
```


If you want the stored procedure to have the same name as those in different replicate databases, see “Specifying a nonunique name for a user-defined function” on page 693 for details.

- 7 In the replicate database, use the `sp_setreplicate` system procedure or the `sp_setrepproc` system procedure to mark the stored procedure for replication. For example:

```
sp_setreplicate upd_emp, 'true'
```

or

```
sp_setrepproc upd_emp, 'table'
```

See “Specifying stored procedures and tables for replication” on page 688 for details on using `sp_setreplicate`. See “Marking stored procedures for replication” on page 343 for details on using `sp_setrepproc`.

- 8 Create a stored procedure on the primary database with the same name as the stored procedure on the replicate database, but do not mark the procedure as replicated. This stored procedure modifies a primary table. For example:

```
create proc upd_emp
  @emp_id int, @salary float
as
update employee
set salary = salary * @salary
where emp_id = @emp_id
```

Note The stored procedure names on the primary and replicate databases can differ if you alter the function string for the function to execute a stored procedure with a different name. See “Mapping to a different stored procedure name” on page 691 for more information.

- 9 Grant permission on the stored procedure to the replicate Replication Server users who will execute this stored procedure. For example:

```
grant all on upd_emp to public
```
- 10 Create a user-defined function on the primary Replication Server that associates the stored procedure to the name of a replication definition for the table it updates. For example:

```
create function employee_rep.upd_emp
  (@emp_id int, @salary float)
```
- 11 Verify that all Replication Server and database objects in steps 1 through 10 exist at the appropriate locations.

Refer to Chapter 6, “Adaptive Server Stored Procedures,” in the *Replication Server Reference Manual* for information about stored procedures used to query the RSSD for system information.

Specifying stored procedures and tables for replication

You can use the `sp_setreplicate` system procedure in Adaptive Server to mark database tables and stored procedures for replication.

You can also use the `sp_setreptable` system procedure to mark tables for replication and the `sp_setrepproc` system procedure to mark stored procedures for replication. These system procedures extend the capabilities of `sp_setreplicate` and are intended to replace it.

See “Using the `sp_setreptable` system procedure” on page 276 and “Marking stored procedures for replication” on page 343 for details.

The syntax for the `sp_setreplicate` system procedure is:

```
sp_setreplicate [object_name [, { 'true' | 'false' } ]]
```

object_name can be either a table name or a stored procedure name.

The “true” and “false” parameters change the replication status of a specified object. (The single quotes are optional.)

- Use `sp_setreplicate` with no parameters to list all replicated objects in the database.
- Use `sp_setreplicate` with just the object name to check the replication status of the object. Adaptive Server reports 'true' if replication is enabled for the object, or 'false' if it is not.
- Use `sp_setreplicate` with the object name and either 'true' or 'false' to enable or disable replication for the object. You must be the Adaptive Server System Administrator or the Database Owner to use `sp_setreplicate` to change the replication status of an object.

Warning! A replicated stored procedure should only modify data in the database in which it is executed. If it modifies data in another database, Replication Server replicates the updated data and the stored procedure.

Managing user-defined functions

This section describes commands for managing user-defined functions. See Chapter 8, “Managing Replication Server Security” for a list of permissions that are required to use the commands. See Chapter 14, “Customizing Database Operations” for details on altering function strings for user-defined functions and displaying function-related information.

Creating a user-defined function

Use the create function command to register a replicated stored procedure with Replication Server. When a stored procedure is executed, Replication Server maps it to a replication definition. The replication definition contains a user-defined function name that matches the name of the stored procedure.

Replication Server delivers the function to the Replication Server that is primary for the replication definition. When the destination Replication Server that owns the replication definition receives the function, it maps the stored procedure parameters into the commands for the user-defined function.

The syntax for the create function command is:

```
create function replication_definition.function
([@parameter datatype [, @parameter datatype]...])
```

The *replication_definition* must be an existing replication definition.

Observe these guidelines when using this command:

- Execute this command at the Replication Server where the replication definition was created.
- Do not use the names of system functions. See Chapter 14, “Customizing Database Operations” for the list of reserved system-function names.
- Include the parentheses surrounding the listed parameters, even when you are defining functions with no parameters.
- If you are not using a function-string class for which default generated function strings are provided, after you have created a user-defined function, use the create function string command to add a function string. See Chapter 14, “Customizing Database Operations” for details.

The following example creates a user-defined function named `Stock_receipt`. The function is associated with the `Items_rd` replication definition:

```
create function Items_rd.Stock_receipt
```

```
(@Location int, @Recpt_num int,  
@Item_no char(15), @Qty_recd int)
```

When a user executes the replicated stored procedure, Replication Server now delivers it.

Adding parameters to a user-defined function

When you add a parameter to a replicated stored procedure, use the alter function command to tell Replication Server about the new parameters. To add the parameters:

- 1 Alter the stored procedure at the primary or replicate data server and provide defaults for new parameters.
- 2 As a precaution, quiesce the system. Altering functions while updates are in process can have unpredictable results.

See “Quiescing Replication Server” on page 111 for details on quiescing the system.

- 3 Alter the function using the alter function command.
- 4 If you are not using a function-string class for which default generated function strings are provided, alter function strings to use the new parameters. See Chapter 14, “Customizing Database Operations” for details.

The syntax for the alter function command is:

```
alter function replication_definition.function  
add parameters @parameter datatype  
[, @parameter datatype]..
```

The *replication_definition* is the name of the replication definition for the function. A function can have up to 255 parameters.

The following example adds an int parameter named Volume to the New_issue function for the Tokyo_quotes replication definition:

```
alter function Tokyo_quotes.New_issue  
add parameters @Volume int
```

Dropping a user-defined function

Use the drop function command to drop a user-defined function. This command drops a function name and any function strings that have been created for it. You cannot drop system functions.

Before you drop the user-defined function, be sure to:

- 1 Drop the stored procedure at the primary database using the drop procedure Adaptive Server command, or use the `sp_setrepl` or `sp_setreproc` system procedure and specify 'false' to disable replication for the stored procedure.

See “Specifying stored procedures and tables for replication” on page 688 for details on using `sp_setrepl`. See “Marking stored procedures for replication” on page 343 for details on using `sp_setreproc`.

- 2 As a precaution, quiesce the system before executing the drop function command. Dropping functions while updates are in process can have unpredictable results.

See “Quiescing a replication system” on page 111 for details on quiescing the system.

The syntax for the drop function command is:

```
drop function replication_definition.function
```

Execute the command on the Replication Server where the replication definition was created.

The following command drops the `Stock_receipt` user-defined function created in the previous section:

```
drop function Items_rd.Stock_receipt
```

Mapping to a different stored procedure name

When you create a user-defined function in a database that uses the a function-string class for which default generated function strings are provided, Replication Server generates a default function string. The default generated function string executes a stored procedure with the same name and parameters as the user-defined function.

For example, if you are using a default function string, you can set up a request stored procedure to execute in the replicate database by creating a stored procedure in the primary database with the same name and parameters as the user-defined function.

If you want to map the user-defined function to a different stored procedure name, use the alter function string command to configure Replication Server to deliver the stored procedure by executing a stored procedure with a different name. You can also do so in function-string classes that allow you to customize function strings.

Example

This example illustrates how to map a user-defined function to a different stored procedure name.

- 1 Assume the stored procedure `upd_sales` exists on the primary Adaptive Server, and that it performs an update on the Adaptive Server sales table:

```
create proc upd_sales
  @stor_id varchar(10),
  @ord_num varchar(10),
  @date datetime
as
update sales set date = @date
where stor_id = @stor_id
and ord_num = @ord_num
```

- 2 To register the `upd_sales` stored procedure with the Replication Server, create the following function, whose name includes in its name the `sales_def` replication definition on the sales table and the `upd_sales` replicated stored procedure:

```
create function sales_def.upd_sales
  (@stor_id varchar(10), @date datetime)
```

- 3 On the replicate Adaptive Server, a version of the stored procedure `upd_sales` that performs no work is created with the same name:

```
create proc upd_sales
  @stor_id varchar(10),
  @ord_num varchar(10),
  @date datetime
as
print "Attempting to Update Sales Table"
print "Processing Update Asynchronously"
```

- 4 To execute the `upd_sales` stored procedure with the name `real_update` instead of `upd_sales`:
 - The default generated function string is altered:

```
alter function string sales_def.upd_sales
for rs_sqlserver_function_class
output rpc
'execute real_update
@stor_id = ?stor_id!param?,
@date = ?date!param?'
```

- A stored procedure in the primary database is created with the name `real_update`. It accepts two parameters.

Specifying a nonunique name for a user-defined function

The name of a user-defined function must be globally unique in the replication system so that Replication Server can locate the particular replication definition for which the user-defined function is defined. If you create more than one replication definition for the same primary table, there is only one user-defined function for all of that table's replication definitions.

If the user-defined function name is not unique, the first parameter of the stored procedure must be `@rs_repdef`, and the name of the replication definition must be passed in this parameter when the stored procedure is executed.

Do not define the `@rs_repdef` parameter in the create function command for the user-defined function. The replication agent extracts the replication definition name and sends it with the LTL commands. This convention works with RepAgent for Adaptive Server or LTM for SQL Server, but may not be supported by replication agents for other data servers.

Example

This example assumes that the user-defined function is not unique and the replication definition name is passed to the `@rs_repdef` parameter when the following stored procedure is executed:

```
create proc upd_sales
@rs_repdef varchar(30),
@stor_id varchar(10),
@date datetime
as
print "Attempting to Update Sales Table"
print "Processing Update Asynchronously"
```


LTM for SQL Server

This appendix describes the Log Transfer Manager (LTM), which is the replication agent for SQL Server databases.

Note LTM is the replication agent for SQL Server version 11.0.x and earlier. LTM is not used with Adaptive Server version 11.5 and later.

Name	Page
Overview	696
Data flow for replication systems with LTMs	696
SQL Server LTM executable program	701
Configuring and maintaining the LTM	702
Suspending and resuming log transfer	708
Modifying replication systems with LTMs	709
LTM error log information	712

This appendix describes LTM and its function in the replication system. It describes how to configure and maintain it and provides procedures for changing the status of a database from primary to replicate and vice versa. It describes how the LTM logs errors.

For detailed information about these topics and for information about warm standby applications and recovery operations when your replication system includes LTMs, see the *Replication Server Administration Guide* for version 11.0.

Overview



LTM is the replication agent for SQL Server databases. LTM notifies Replication Server of actions in a database that must be replicated to other databases. LTM reads the database transaction log and transfers log records for replicated tables and stored procedures to the Replication Server managing the database. The Replication Server distributes the modifications to sites with subscriptions for the data.

An LTM is needed for every database that contains primary data and for every database where asynchronous procedures are executed. A database that contains only copies of replicated data and contains no asynchronous procedures does not require an LTM.

See “LTM processing data flow” on page 697 for a description of how LTM reads transaction information in the SQL Server, formats it, and submits it to Replication Server.

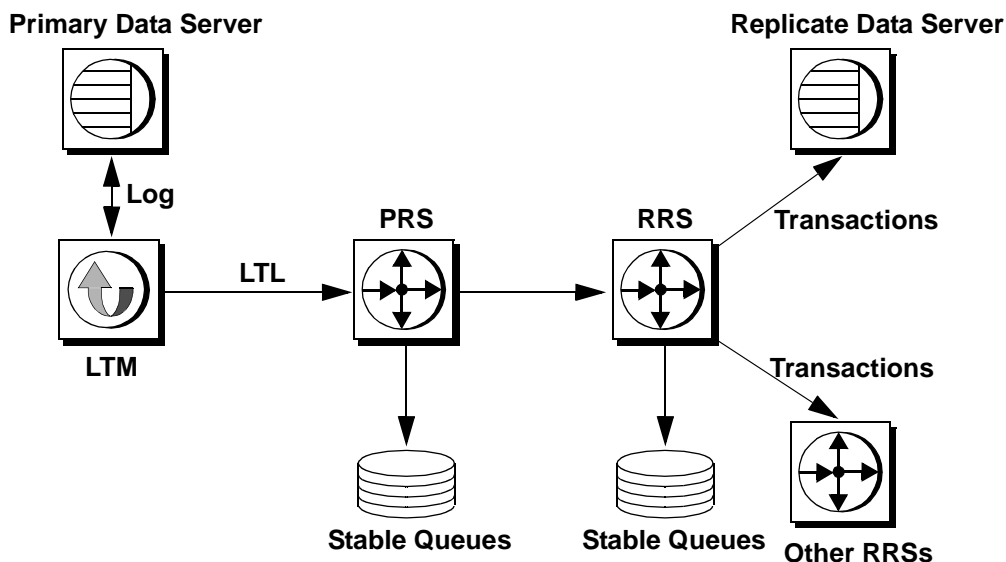
Data flow for replication systems with LTMs

This section assumes a typical scenario where a Replication Server exists at both primary and replicate sites, as shown in Figure B-1. Each time a client requests a transaction that changes database content, the following actions occur:

- 1 The LTM reads the log of the primary database for transactions that are marked for replication.
- 2 The LTM forwards the transactions, through the Log Transfer Language (LTL), to the primary Replication Server (PRS in the figure), where they are stored in a stable queue.
- 3 The primary Replication Server determines, for each transaction, which of the following actions to take:
 - Discard the transaction if no subscriptions exist for the data.
 - Forward the transaction to each replicate Replication Server (RRS in the figure), where it is stored in a stable queue.

- 4 The replicate Replication Server determines, for each transaction, which of the following actions to take:
 - Route the transaction to another Replication Server.
 - Applies the transaction to replicate databases that it manages.

Figure B-1: Replication Server overview



LTM processing

The LTM for SQL Server is an Open Server application. It connects to a primary database as a client, retrieves the data for replicated objects from the SQL Server log, and converts the log record information into LTL commands. These commands are then sent to the primary Replication Server for distribution and replication.

LTM processing data flow

The following describes the flow of data in LTM processing. The LTM performs the following actions, as shown in Figure B-2.

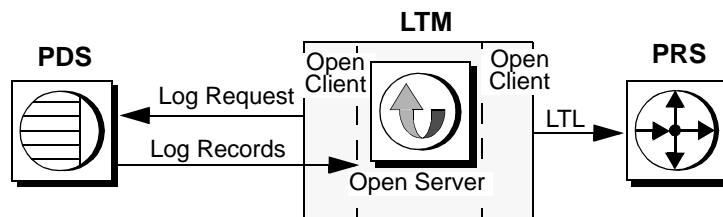
- 1 Scans the primary database log and forwards transaction log records to the primary Replication Server (PRS in the figure) involving tables that are marked for replication.
- 2 Filters out records written by the database maintenance user.

This step prevents transactions that are replicated to the primary database from being re-replicated out of the database to other sites.

Note For applications designed using the redistributed corporate rollup model, LTM must be started with the -A flag, which allows replicated transactions to be redistributed as primary data. See “Redistributed corporate rollup” on page 14 for more information.

- 3 Marks the transaction log so the primary data server (PDS in the figure) does not truncate log records that have not yet been passed for replication. For details, see “Data server log truncation” on page 698

Figure B-2: LTM processing



Data server log truncation

The LTM retrieves transactions from the SQL Server log and sends them to the primary Replication Server. As long as there is space in the database log, the data server can continue to process updates. To prevent the log from filling up, you need to truncate it using the SQL Server dump transaction command.

SQL Server and the LTM cooperate to ensure that only transactions already processed by the LTM and passed to Replication Server are truncated.

The LTM maintains a secondary truncation point in the SQL Server log. The secondary truncation point is the log page that contains the begin transaction command for the oldest transaction not yet fully received by Replication Server. The LTM resets the truncation point when SQL Server returns an end-of-scan message. Two conditions cause SQL Server to return an end-of-scan message:

- When the number of log records returned to the LTM has reached the number specified with the *batch_sz* configuration parameter
- When the time interval specified by the *scan_retry* configuration parameter has expired

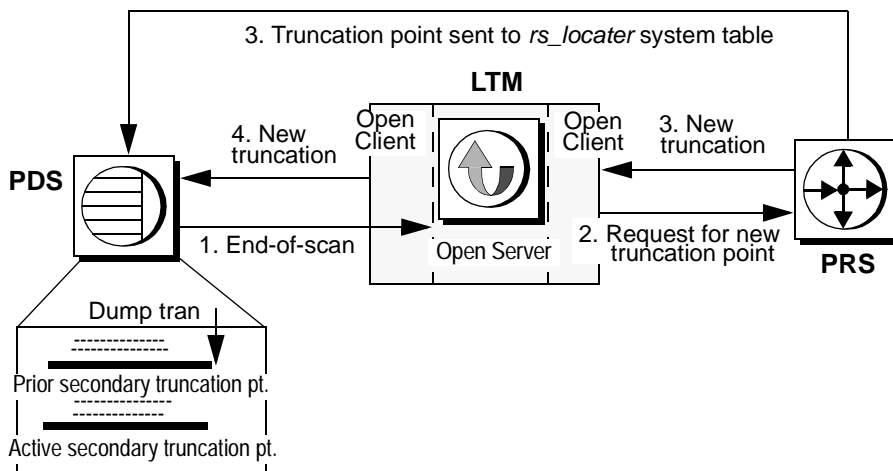
See “LTM configuration file” on page 705 for information about setting these parameters in the LTM configuration file.

The *batch_sz* condition

The *batch_sz* configuration parameter is the number of log records to request from the SQL Server in each batch before moving the LTM truncation point. The following events result in the LTM setting the truncation point, as shown in Figure B-3 on page 699:

- 1 When the primary data server (PDS in the figure) reaches the *batch_sz* limit, it sends an end-of-scan message to the LTM.
- 2 When the LTM receives an end-of-scan message, it requests a new truncation point from the primary Replication Server (PRS in the figure).
- 3 The primary Replication Server returns the latest truncation point to the LTM and writes it to the *rs_locator* system table.
- 4 The LTM executes the *dbcc settrunc* command in the primary data server to set the LTM truncation point.

Figure B-3: Data server log truncation



The *scan_retry* condition

When there are no log records, the data server log scan thread sleeps. New log activity awakens the log scan thread. The *scan_retry* configuration parameter limits the length of time to sleep. When the data server reaches the maximum scan retry time, and there is still no new log activity, it sends an end-of-scan message to the LTM. Processing occurs as in steps 2 through 4 in the preceding section.

When new records are added to the end of the log, the log scan thread wakes up and sends the new records. If *batch_sz* is not reached yet, it goes back to sleep. The time remaining in *scan_retry* is added to the next sleep interval, for a maximum time of *scan_retry**2 seconds. When the log scan thread sleep interval has expired, the thread returns an end-of-scan message to the LTM, and a new log scan request is initiated.

Note If you restart an LTM while the log scan thread is sleeping, the SQL Server rejects the LTM connection because the sleeping log scan thread has not released the previous LTM connection. When the thread wakes up, the SQL Server releases the previous connection and can then accept a new LTM connection. If you create a script to automatically restart an LTM, you should have the script sleep for *scan_retry* * 2 seconds before it restarts the LTM.

LTM user thread

There is one LTM user thread for each primary database that the Replication Server manages. The LTM user thread manages an LTM connection. It verifies that LTM submissions are valid and writes them into the inbound stable queue for the database.

If the connection between an LTM and a Replication Server is broken, the LTM user thread shuts down.

You can monitor general information on current Replication Server threads by using the command `admin who`. On the display output, the LTM user thread appears as “LTM USER”.

SQL Server LTM executable program

If the Replication Server has an LTM for a SQL Server database, use the `ltm` operating system command to start the LTM. Do this after the RSSD SQL Server and the Replication Server are already running. Here is the syntax for `ltm`:

```
ltm [-C config_file] [-S ltm_name]  
    [-I interfaces_file] [-E errorlog_file]  
    [-M] [-A] [-W] [-v]
```

Refer to “`ltm`” in Chapter 7, “Programs,” in the *Replication Server Reference Manual* for complete information about each of the parameters of the `ltm` command.

The `rs_init` program creates the run file “`RUN_name`”, where *name* is the name of the LTM. The run file specifies the `ltm` command with parameters set for the installed LTM. Normally, you start LTM by executing this file.

The `ltm` executable program and the LTM run file are 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.

Warning! If you are running more than one LTM, either execute them in separate directories, or use the `-E` flag to specify different error log file names. Otherwise, the LTMs will try to record their messages in the same file.

Shutting down an LTM

To shut down an LTM, log in to it and enter this at the `isql` prompt:

```
shutdown
```

When you shut down an LTM, it refuses additional connections, terminates threads, and exits.

Checking log files for errors

Like the Replication Server, the LTM displays status and error messages in its log file. The default name for the log file is *ltm.log*. You can change this default by restarting `ltm` with the `-E` option and specifying the error log file name to use.

You can check the *lrm.log* files for any error messages. One way to do this is by using Sybase Central, which also lets you invoke shell scripts based on errors reported in those logs.

RSM

In Sybase Central, see the topics under “Viewing the Error Log” in Replication Server’s plug-in help.

Messages continue to accumulate in the error log files until you remove them. For this reason, you may choose to truncate the log files when the LTM is shut down.

If a log file is unavailable, important error information is written to the standard error output file, which you can display on a terminal or redirect to a file.

Configuring and maintaining the LTM

An LTM for the RSSD is installed with the Replication Server if the Replication Server is the source site for any route. In addition, a database in a replication system requires an LTM if:

- The database holds primary data, or
- Replicated stored procedures are executed in the database

The following sections describe how to maintain LTMs for the RSSD and the databases in your system.

The only task you may need to perform for the RSSD LTM is modifying the configuration file. An LTM configuration file contains information the LTM needs to find the database log it is transferring and the Replication Server to which it is transferring the log.

This section describes the components and resources that must be present or assembled before you can run LTM. See the *Replication Server Design Guide* for component requirements for different replication topologies and performance considerations when constructing topologies.

Adding a Replication Server

To add a Replication Server, use the `rs_init` installation program, as described in the *Replication Server Installation Guide* for your platform. When connecting a new Replication Server to an existing system, always conduct a careful review and analysis of how the server will fit into your system. Determine what other processes are required for the server and designate required names and accounts for these processes.

RSM

You can also add a Replication Server using Replication Server plug-in. See the topics under, “Adding a server to an RSM Server domain” in Replication Server plug-in help for more information.

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

- Creates configuration files for the Replication Server and the RSSD LTM
- Creates executable files to start the Replication Server and the RSSD LTM
- Sets up the RSSD
- Starts the Replication Server and the RSSD LTM

Preparing databases for replication

SQL Server databases are prepared for use with Replication Server using the `rs_init` program described in the Replication Server installation and configuration guides for your platform.

rs_init installation program

The `rs_init` program prepares a database for replication. If the database has primary data, `rs_init`:

- Creates an LTM configuration file
- Creates a run file, an executable script to start the LTM
- Starts the LTM
- Sets the secondary truncation point to “valid” in the SQL Server database, preventing SQL Server from truncating database log records before the LTM has retrieved them

Refer to the Replication Server installation and configuration guides for your platform for details on each step.

If you are adding a database that requires an LTM, you specify the following information in `rs_init`:

- LTM name
- Replication Server user and password
- LTM Administrator user and password
- LTM error log file name
- LTM configuration file name
- LTM password encryption
- LTM character set
- LTM language
- LTM sort order
- LTM interfaces file information

Interfaces file

The interfaces file contains network definitions for servers in the replication system, including Replication Servers, data servers, and LTMs. Server programs are registered in an interfaces file so that client applications and other server programs can locate them.

Generally, one interfaces file at each site contains entries for all local and remote Replication Servers and data servers. The interfaces file at a site requires entries for these LTM components:

- RSSD LTM for this Replication Server
- LTMs for databases managed by this Replication Server

You can use the default interfaces file or you can specify an alternative interfaces file at the command line when you start Replication Server or LTM. The interfaces file is usually located in the SYBASE release directory. Use `ds_edit` to modify the interfaces file. See the Replication Server installation guide for your platform for more information.

Note With Replication Server 11.5 and later, if you are using network-based security, we recommended that you use the directory services of your network security mechanism to register Replication Servers, SQL Servers, and gateway software. For details, see the documentation included with your network security mechanism.

LTM configuration file

The configuration file for an LTM for a primary database or an RSSD is created during the installation process. If you are modifying a primary database, you may need to modify the configuration file for the LTM for the database. Refer to the Replication Server installation and configuration guides for your platform.

LTM finds the start-up information it needs in a configuration file. The format of the LTM configuration file is the same as the Replication Server configuration file, which is described in “Setting Replication Server configuration parameters” on page 99.

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`. Refer to the Replication Server installation and configuration guides for your platform for more information about working with encrypted passwords. The default name for the LTM configuration file is the LTM name with “.cfg” appended.

Note If a password is stored in encrypted form, you cannot edit the password in the LTM configuration file. To change an encrypted password in this file, use the `rs_init` installation program. See the Replication Server installation guide for your platform for more information.

Refer to the reference page for “lrm” in Chapter 7, “Programs,” in the *Replication Server Reference Manual* for detailed descriptions of LTM configuration file parameters.

Replication Server login name and password for the LTM

The LTM retrieves information about changes to the replicated system tables from the database transaction logs and submits them to the Replication Server for distribution.

Replication Server needs a login name for LTM use. 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 LTM. See the *Replication Server Reference Manual* for command syntax details.

- To change the login name and/or password (encrypted or clear text), use the `alter user` command with the `set password` clause.

This updates the login and/or password in the `rs_users` system table.

- To change both the login name and password (encrypted or clear text), 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.
- Update the LTM configuration file with the new login name and/or password.
- For the updates to take effect, restart the LTM.

LTM login name and password

The LTM configuration file contains the login name and password for the System Administrator LTM user. Use this login name and password to log in to and shut down the LTM. The configuration parameters that hold the login name and password are `LTM_admin_user` and either `LTM_admin_pw` or `LTM_admin_pw_enc`.

By default, the System Administrator LTM user's login name is "sa" and the password is "null." If the login name or password is changed, edit the LTM configuration file to match the changes and restart the LTM.

SQL Server login name and password

LTM log in to SQL Server to read the transaction log by using either the “sa” login name or the “dbo” role, depending on the values you supply when you use `rs_init`. If you change the password for the “sa” login name or the “dbo” role, you must also change it in the LTM configuration file. Be sure to shut down and restart the LTM so that the new values can take effect.

Enabling and disabling password encryption

When you use the `rs_init` program to install or upgrade the replication system, you can enable password encryption. This allows `rs_init` to encrypt passwords throughout sensitive areas of the replication system. After the replication system is installed or upgraded, you can use `rs_init` at any time to enable encryption.

If encryption is enabled for a Replication Server, `rs_init` encrypts new passwords, passwords contained in the Replication Server configuration file, and passwords stored in the RSSD. If encryption is enabled for an LTM, `rs_init` encrypts new passwords and passwords contained in the LTM configuration file.

For details on enabling password encryption using the `rs_init` program, see the *Replication Server Installation Guide*.

Disabling encryption on new and existing LTM passwords

Once LTM password encryption is enabled through `rs_init`, use this procedure to decrypt LTM passwords:

- 1 Disable encryption on new passwords that are entered for LTM.
To do this, specify 0 as the value of the `pwd_encrypt` configuration parameter in the LTM configuration file.
- 2 Restart the LTM to pick up the new `pwd_encrypt` configuration parameter.

Suspending and resuming log transfer

If you are performing recovery, troubleshooting, or diagnostic tasks, you may need to suspend and resume log transfer. This is described in the following section. See Chapter 14, “Replication System Recovery,” in the *Replication Server Administration Guide* for version 11.0 for procedures to start the LTM in recovery mode so that it can replay database and transaction dumps.

Suspending LTMs

To disconnect an LTM and prevent an LTM from connecting to the Replication Server, execute the suspend log transfer command. The LTM remains suspended until you restart the LTM and execute the resume log transfer command.

Note Suspending LTMs is the first step in quiescing the replication system. See “Quiescing Replication Server” on page 111.

The syntax for the suspend log transfer command is:

```
suspend log transfer from {data_server.database | all}
```

data_server – the data server with the database whose LTM is to be suspended.

database – the database whose LTM is to be suspended and whose connections are to be disallowed.

all – instructs Replication Server to suspend all LTMs and disallow future connections for all LTMs.

The suspend log transfer command records information in the RSSD, so suspended LTMs remain suspended after the Replication Server is restarted.

Examples

The following examples demonstrate the use of the suspend log transfer command.

- 1 The following command disconnects the LTM for the database named NY_ACCOUNTS_DB:

```
suspend log transfer from NY_DS.NY_ACCOUNTS_DB
```

- 2 The following command suspends all LTM connections to this Replication Server:

```
suspend log transfer from all
```

In both examples, after the command is executed, the LTM process shuts down.

Resuming LTMs

This command allows an LTM to connect to a Replication Server. However, resume log transfer does not restart the LTM. The LTM must also be restarted manually.

The syntax for the resume log transfer command is:

```
resume log transfer from {data_server.database | all}
```

data_server – the data server with the database whose LTM is to be resumed.

database – the database whose LTM is to be resumed.

all – allow all LTMs to connect to this Replication Server.

Examples

These examples demonstrate the use of the resume log transfer command.

- 1 The following command allows the LTM for NY_ACCOUNTS_DB to connect to the Replication Server:

```
resume log transfer from NY_DS.NY_ACCOUNTS_DB
```

- 2 The following command allows all LTMs to connect to the Replication Server:

```
resume log transfer from all
```

In both examples, the appropriate LTMs must then be restarted.

Modifying replication systems with LTMs

The following sections describe how you can:

- Configure Replication Servers to manage primary tables
- Change replicate to primary databases
- Change primary to replicate databases

when your system uses LTMs as replication agents.

Configuring Replication Servers to manage primary tables

If you want to add a route from a Replication Server that used to be configured as a replicate-only Replication Server, the RSSD for that Replication Server requires an LTM.

To add an LTM for the RSSD, perform the following steps:

- 1 To set the LTM truncation point for the RSSD to “valid,” log in to the SQL Server as “sa” and execute the following commands:

```
use RSSD_database
go
dbcc settrunc('ltm,' 'valid')
go
```

- 2 To allow the Replication Server to receive messages from the LTM, execute the following command at the Replication Server:

```
alter connection to RSSD_data_server.RSSD_database
set log transfer on
```

- 3 Add an interfaces file entry for the LTM.

Use *ds_edit* to modify the interfaces file. See the *Replication Server Installation Guide* for more information.

- 4 Create an LTM configuration file.
- 5 Run the LTM for the configuration file.

Changing replicate databases to primary databases

This section describes how to change a database that is designated as “replicate only” to be a source of asynchronous transactions or to contain primary data.

- 1 At the Replication Server managing the database, create *RS_user* so that the LTM can log in to the Replication Server.

Use the create user command:

```
create user RS_user_name
set password {RS_password | null}
```

where *RS_user_name* and *RS_password* are the name and password the LTM uses to connect to Replication Server.

Grant this user connect source permission, using the grant command:

```
grant connect source to RS_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.

- 2 At the SQL Server, create SQL_user so that the LTM can log in to the SQL Server. Grant SQL_user dbo permission or replication_role.

See the SYBASE *SQL Server System Administration Guide* for information about creating users and granting permissions in SQL Server.

- 3 At the Replication Server that manages the database, execute the alter connection command using the log transfer on option:

```
alter connection to data_server.database
set log transfer on
```

- 4 At the SQL Server, set the LTM truncation point for the database to “valid”:

```
use database
go
dbcc settrunc('ltm', 'valid')
go
```

- 5 Add an interfaces file entry for the LTM that will read the SQL Server log of this new primary database.

Use dsedit to modify the interfaces file. See the installation guide for your platform for more information.

- 6 In the SQL Server, create the rs_marker stored procedure and set its replicate status to “true,” using the sp_setrepproc 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.

- 7 Create an LTM configuration file and start the LTM.

Changing primary databases to replicate databases

Use the following procedure to change a primary database to a replicate database:

- 1 Drop all subscriptions to the replication definitions in this database.
- 2 Drop all replication definitions and functions defined for this database.
- 3 Shut down the LTM.
- 4 Log in to the Replication Server that manages the database and execute the alter connection command using the log transfer off option:

```
alter connection to data_server.database  
set log transfer off
```

- 5 Log in to the SQL Server and set the LTM truncation point for the database to “ignore”:

```
use database  
go  
dbcc settrunc('ltm', 'ignore')  
go
```

- 6 Set the status of rs_marker to “false” using the sp_setrepproc system procedure.

```
sp_setrepproc rs_marker, 'false'
```

- 7 Set the replicate status of all replicated objects to “false”. To do this:
 - a Execute sp_setreptable and sp_setrepproc without any arguments to generate a list of all replicated tables and stored procedures in the database.
 - b One by one, set the replicate status of each table and stored procedure to “false,” using sp_setreptable and sp_setrepproc.

LTM error log information

The SQL Server LTM error log is a text file. The LTM records errors and informational messages that occur when transferring replicated objects from the SQL Server log and converting them into LTL commands. These commands are sent to the Replication Server for distribution and replication. Errors include those from the SQL Server, the Replication Server, or internally from the LTM.

By default, the LTM error log file name is *ltm.log* and resides in the directory where you started the LTM. You can specify the name and location of the error log file by using the -E command line flag when you start the LTM or in an LTM run file

For solutions to common LTM errors, refer to the *Replication Server Troubleshooting Guide* for Replication Server 11.0.x.

The LTM performs actions based on the severity and recoverability of an error. The actions are listed in Table B-1.

Table B-1: Action for LTM errors

Action	Description
Log the error as a warning and continue processing.	This action is taken if the LTM can continue its normal operation, and the error condition does not affect the general integrity of the system. For warning message examples, see the <i>Replication Server Troubleshooting Guide</i> for Replication Server 11.0.x.
Retry the operation that caused the error until it succeeds.	The LTM retries the operation that caused the error, as in the case where the connection to the Replication Server is down, LTM source is already connected, and SQL Server is out of system alarms, or is in the middle of recovery. Retrying an operation that caused an error temporarily prevents the LTM from continuing. The operation should eventually succeed.
Abort and disconnect from the SQL Server and Replication Server.	The action is taken for fatal errors that are too severe to continue. The error cannot be recovered until some corrective action has been taken. The LTM is shut down.

Note Unlike Replication Server, LTM error actions are not user-configurable.

The format of LTM messages is identical to the format for Replication Server messages. For details, see “Error and warning messages” on page 611.

LTM message types

Table B-2 describes the kinds of errors you can expect to find in the LTM error log. For specific errors and how to resolve them, see the *Replication Server Troubleshooting Guide* for Replication Server 11.0.x.

Table B-2: Types of LTM error messages

Component	Description of error messages
SQL Server	Commonly identified by error numbers 9100 through 9199. The LTM can also obtain SQL Server errors outside this range. Error numbers 9100 through 9199 are usually related to scanning the log and are not recorded in the SQL Server log because they are not considered severe to the SQL Server. They are only sent to the client. Look at the LTM error log for errors or conditions that occur during the log scan. The LTM logs the errors and performs some action based on the error. For a list of warning errors in this range, see Table B-2 on page 713. It describes the kinds of errors you can expect to find in the LTM error log. For specific errors and how to resolve them, see the <i>Replication Server Troubleshooting Guide</i> for Replication Server 11.0.x.

Component	Description of error messages
Replication Server	<p>These include Replication Server normalization errors (numbers 32000 through 32999), rebuild or restart errors (numbers 14034 through 14036, and 14069), Open Server errors, and <i>ct_lib</i> and <i>cs_lib</i> errors from libraries the LTM uses.</p> <p>If the LTM is in recovery mode and the error number is 14048, the LTM rebuilds the messages from the SQL Server log.</p> <p>The LTM treats normalization errors as recoverable errors. The LTM logs the error and continues processing. For Open Server and library errors, responses to errors are based on the severity of the error and are handled in a manner similar to the way Replication Server handles library errors. Replication Server shuts down the LTM if the error is fatal.</p> <p>Normalization errors result from transient inconsistencies in the setup of replicated objects. For example, if a table has been marked replicated with the <i>sp_setreptable</i> system procedure, but no replication definitions have yet been created for the table, the LTM will retrieve log records for an object that is not yet known to the Replication Server.</p>
LTM	<p>General LTM errors include running out of memory and other software errors.</p> <p>If the LTM encounters other errors sending the distribute commands (SQL-like statements), it turns off the <i>batch_ltl_cmds</i> configuration option and sends the distribute commands one at a time. If the <i>skip_ltl_cmd_err</i> configuration option is “on,” the LTM skips the command. Otherwise, the LTM shuts down. See “ltm” in Chapter 7, “Programs,” of the <i>Replication Server Reference Manual</i> for details on the configuration settings.</p>

LTM warning messages from SQL Server

The following SQL Server errors from the 9100 and 9199 error range are logged as warnings; processing continues. The messages are printed in the language specified in the *LTM_language* configuration parameter (assuming SQL Server has that language installed).

```
9102 - Failed to convert the log record into row format
for database 'tokyo_ds.pubs2', XACT ID 0x10ab RID 0x04.
Information associated with the log record is not
replicated.
```

```
9103 - Failed to send the log record for database 'xxx',
XACT ID 0x10ab, RID 0x04. Information associated with
the log record is not replicated.
```

```
9105 - A missing log record indicates a prematurely
truncated log or a corrupt log. The log record in
database 'tokyo_ds.pubs2', XACT ID 0x10ab, is not
replicated.
```

```
9106 - The deferred insert (INOOP) log record referenced
by the insert indirect (INSIND) log record was not found
as expected at RID 0x04.
```

```
9107 - Unexpected function return value while processing
the log record of database 'tokyo_ds.pubs2', XACT ID
```

0x10ab, RID 0x04. The log record may not be replicated.9110 - Found an ENDXACT log record before finding an expected INSERT log record in database 'tokyo_ds.pubs2', XACT ID 0x10ab, RID 0x04.
9141 - The stored proc. 'a' associated with the log record in database 'tokyo_ds.pubs2', XACT ID 0x10ab, RID 0x04, was dropped after the log record was written. The log record is not replicated.

High Availability on Sun Cluster 2.2

This appendix provides background and procedures for configuring Sybase Replication Server for high availability (HA) on Sun Cluster 2.2.

Topic	Page
Introduction	717
Terminology	718
Technology overview	719
Configuring Replication Server for high availability	720
Administering Replication Server as a data service	725

Introduction

This appendix assumes that:

- You are familiar with Sybase Replication Server. This chapter does not explain the steps necessary to install Sybase Replication Server.
- You are familiar with Sun Cluster HA. This document does not explain the steps necessary to install Sun Cluster HA.
- You have a two-node cluster hardware with Sun Cluster HA 2.2.

Documentation references:

- *Sun Cluster 2.2 Software Planning and Installation Guide*
- *Sun Cluster 2.2 System Administration Guide*
- *Configuring Sybase Adaptive Server Enterprise 12.0 Server for High Availability: Sun Cluster HA* (see White Papers at <http://www.sybase.com/products/databaseservers/ase>).
- Replication Server documentation (see Product Manuals at <http://www.sybase.com/products/eaimiddleware/replicationserver>).

Terminology

These terms are used in this chapter:

- Cluster – multiple systems, or nodes, that work together as a single entity to provide applications, system resources, and data to users.
- Cluster node – a physical machine that is part of a Sun Cluster. Also called a physical host.
- Data service – an application that provides client service on a network and implements read and write access to disk-based data. Replication Server and Adaptive Server Enterprise are examples of data services.
- Disk group – a well-defined group of multihost disks that move as a unit between two servers in an HA configuration.
- Fault monitor – a daemon that probes data services.
- High availability (HA) – very low downtime. Computer systems that provide HA usually provide 99.999% availability, or roughly five minutes unscheduled downtime per year.
- Logical host – a group of resources including a disk group, logical host name, and logical IP address. A logical host resides on (or is mastered by) a physical host (or node) in a cluster machine. It can move as a unit between physical hosts on a cluster.
- Master – the node with exclusive read and write access to the disk group that has the logical address mapped to its Ethernet address. The current master of the logical host runs the logical host's data services.
- Multihost disk – a disk configured for potential accessibility from multiple nodes.
- Failover – the event triggered by a node or a data service failure, in which logical hosts and the data services on the logical hosts move to another node.
- Failback – a planned event, where a logical host and its data services are moved back to the original hosts.

Technology overview

Sun Cluster HA is a hardware- and software-based high availability solution. It provides high availability support on a cluster machine and automatic data service failover in just a few seconds. It accomplishes this by adding hardware redundancy, software monitoring, and restart capabilities.

Sun Cluster provides cluster management tools for a System Administrator to configure, maintain, and troubleshoot HA installations.

The Sun Cluster configuration tolerates these single-point failures:

- Server hardware failure
- Disk media failure
- Network interface failure
- Server OS failure

When any of these failures occur, HA software fails over logical hosts onto another node and restarts data services on the logical host in the new node.

Sybase Replication Server is implemented as a data service on a logical host on the cluster machine. The HA fault monitor for Replication Server periodically probes Replication Server. If Replication Server is down or hung, the fault monitor attempts to restart Replication Server locally. If Replication Server fails again within a configurable period of time, the fault monitor fails over to the logical host so the Replication Server will be rebooted on the second node.

To Replication Server clients, it appears as though the original Replication Server has experienced a reboot. The fact that it has moved to another physical machine is transparent to the users. Replication Server is affiliated with a logical host, not the physical machine.

As a data service, the Replication Server includes a set of scripts registered with Sun Cluster as callback methods. Sun Cluster calls these methods at different stages of Failover:

- `FM_STOP` – to shut down the fault monitor for the data service to be failed over.
- `STOP_NET` – to shut down the data service itself.
- `START_NET` – to start the data service on the new node.
- `FM_START` – to start the fault monitor on the new node for the data service.

Each Replication Server is registered as a data service using the `hareg` command. If you have multiple Replication Servers running on the cluster, you must register each of them. Each data service has its own fault monitor as a separate process.

Note For detailed information about the `hareg` command, see the appropriate Sun Cluster documentation.

Configuring Replication Server for high availability

This section describes the tasks required to configure a Replication Server for HA on Sun Cluster (assuming a two-node cluster machine).

- “Configuring Sun Cluster for HA” on page 720
- “Installing Replication Server for HA” on page 721
- “Installing Replication Server as a data service” on page 722

Configuring Sun Cluster for HA

The system should have following components:

- Two homogenous Sun Enterprise servers with similar configurations in terms of resources like CPU, memory, and so on. The servers should be configured with cluster interconnect, which is used for maintaining cluster availability, synchronization, and integrity.
- The system should be equipped with a set of multihost disks. The multihost disk holds the data (partitions) for a highly available Replication Server. A node can access data on a multihost disk only when it is a current master of the logical host to which the disk belongs.
- The system should have Sun Cluster HA software installed, with automatic failover capability. The multihost disks should have unique path names across the system.
- For disk failure protection, disk mirroring (not provided by Sybase) should be used.

- Logical hosts should be configured. Replication Server runs on a logical host.
- Make sure the logical host for the Replication Server has enough disk space in its multihomed disk groups for the partitions, and that any potential master for the logical host has enough memory for the Replication Server.

Installing Replication Server for HA

During Replication Server installation, you need to perform these tasks in addition to the tasks described in the Replication Server installation guide:

- 1 As a Sybase user, load Replication Server either on a shared disk or on the local disk. If it is on a shared disk, the release cannot be accessed from both machines concurrently. If it is on a local disk, make sure the release paths are the same for both machines. If they are not the same, use a symbolic link, so they will be the same. For example, if the release is on */node1/repserver* on node1, and */node2/repserver* on node2, link them to */repserver* on both nodes so the *\$SYBASE* environment variable is the same across the system.
- 2 Add entries for Replication Server, RSSD server, and primary/replicate data servers to the interfaces file in the *\$SYBASE* directory on both machines. Use the logical host name for Replication Server in the interfaces file.

Note To use LDAP directory services instead of interfaces files, supply multiple entries in the DIRECTORY section of the Replication Server configuration file. If the connection to the first entry fails, the directory control layer (DCL) attempts to connection to the second entry and so on. If a connection cannot be made to any entry in the DIRECTORY section, Open Client/Server *does not* use the default interfaces file to attempt a connection.

See the configuration guide for your platform for information about setting up LDAP directory services.

- 3 Start the RSSD server.
- 4 Follow the installation guide for your platform to install Replication Server on the node that is currently the master in the logical host. Make sure that you:

- a Set the environment variables SYBASE, SYBASE_REP, and SYBASE_OCS:

```
setenv SYBASE /REPSEVER1210
setenv SYBASE_REP REP-12_1
setenv SYBASE_OCS OCS-12_0
```

/REPSEVER1210 is the release directory.

- b Choose a run directory for the Replication Server that will contain the Replication Server run file, configuration file, and log file. The run directory should exist on both nodes and have exactly the same paths on both nodes (the path can be linked if necessary).
- c Choose the multihosted disks for the Replication Server partitions.
- d Initiate the `rs_init` command, from the *run* directory:

```
cd RUN_DIRECTORY
$SYBASE/$SYBASE_REP/install/rs_init
```

- 5 Make sure that Replication Server is started.
- 6 As a Sybase user, copy the run file and the configuration file to the other node in the same path. Edit the run file on the second node to make sure it contains the correct path of the configuration and log files, especially if links are used.

Note The run file name must be *RUN_repservers_name*, where *repservers_name* is the name of the Replication Server. You can define the configuration and log file names.

Installing Replication Server as a data service

You also need to perform these specialized tasks to install Replication Server as a data service:

- 1 As root, create the directory */opt/SUNWcluster/ha/repservers_name* on both cluster nodes, where *repservers_name* is the name of your Replication Server. Each Replication Server must have its own directory with the server name in the path. Copy the following scripts from the Replication Server installation directory *\$SYBASE/\$SYBASE_REP/sample/ha* to:

/opt/SUNWcluster/ha/repservers_name

on both cluster nodes, where *repserver_name* is the name of your Replication Server:

```
repserver_start_net
repserver_stop_net
repserver_fm_start
repserver_fm_stop
repserver_fm
repserver_shutdown
repserver_notify_admin
```

If the scripts already exist on the local machine as part of another Replication Server data service, you can create the following as a link to the script directory instead:

```
/opt/SUNWcluster/ha/repserver_name
```

- 2 As root, create the directory `/var/opt/repserver` on both nodes if it does not exist.
- 3 As root, create a file `/var/opt/repserver/repserver_name` on both nodes for each Replication Server you want to install as a data service on Sun Cluster, where *repserver_name* is the name of your Replication Server. This file should contain only two lines in the following form with no blank space, and should be readable only by root:

```
repserver:logicalHost:RunFile:releaseDir:SYBASE_OCS
:SYBASE_REP

probeCycle:probeTimeout:restartDelay:login/password
```

where:

- *repserver* – the Replication Server name.
- *logicalHost* – the logical host on which Replication Server runs.
- *RunFile* – the complete path of the runfile.
- *releaseDir* – the \$SYBASE installation directory.
- *SYBASE_OCS* – the \$SYBASE subdirectory where the connectivity library is located.
- *SYBASE_REP* – the \$SYBASE subdirectory where the Replication Server is located.
- *probeCycle* – the number of seconds between the start of two probes by the fault monitor.

- *probeTimeout* – time, in seconds, after which a running Replication Server probe is aborted by the fault monitor, and a timeout condition is set.
- *restartDelay* – minimum time, in seconds, between two Replication Server restarts. If, in less than *restartDelay* seconds after a Replication Server restart, the fault monitor again detects a condition that requires a restart, it triggers a switch over to the other host instead. This resolves situations where a database restart does not solve the problem.
- *login/password* – the login/password the fault monitor uses to ping Replication Server.

To change *probeCycle*, *probeTimeout*, *restartDelay*, or *login/password* for the probe after Replication Server is installed as data service, send SIGINT(2) to the monitor process (*repserver_fm*) to refresh its memory.

```
kill -2 monitor_process_id
```

- 4 As root, create a file */var/opt/repserver/repserver_name.mail* on both nodes, where *repserver_name* is the name of your Replication Server. This file lists the UNIX login names of the Replication Server administrators. The login names should be all in one line, separated by one space.

If the fault monitor encounters any problems that need intervention, this is the list to which it sends mail.

- 5 Register the Replication Server as a data service on Sun Cluster:

```
hareg -r repserver_name \  
-b "/opt/SUNWcluster/ha/repserver_name" \  
-m START_NET="/opt/SUNWcluster/ha/repserver_name/  
repserver_start_net" \  
-t START_NET=60 \  
-m STOP_NET="/opt/SUNWcluster/ha/repserver_name/  
repserver_stop_net" \  
-t STOP_NET=60 \  
-m FM_START="/opt/SUNWcluster/ha/repserver_name/  
repserver_fm_start" \  
-t FM_START=60 \  
-m  
FM_STOP="/opt/SUNWcluster/ha/repserver_name/repse  
er_fm_stop" \  
-t FM_STOP=60 \  
[-d sybase] -h logical_host
```

where *-d sybase* is required if the RSSD is under HA on the same cluster, and *repserver_name* is the name of your Replication Server and must be in the path of the scripts.

- 6 Turn on the data service using `hareg -y repserver_name`.

Administering Replication Server as a data service

This section describes how to start and shut down Replication Server as a data service, and useful logs for monitoring and troubleshooting.

Data service start/shutdown

Once a Replication Server is registered as data service, use the following to start Replication Server as a data service:

```
hareg -y repserver_name
```

This starts Replication Server if it is not already running, and also starts the fault monitor for Replication Server.

To shut down Replication Server, use:

```
hareg -n repserver_name
```

The fault monitor restarts or fails over this Replication Server if it is shut down or stopped (killed) any other way.

Logs

There are several logs you can use for debugging:

- Replication Server log – the Replication Server logs its messages here. Use the log to find informational and error messages from Replication Server. The log is located in the Replication Server *Run* directory.
- Script log – the data service START and STOP scripts log messages here. Use the log to find informational and error messages that result from running the scripts. The log is located in */var/opt/repserver/harep.log*.

- Console log – the operating system logs messages here. Use this log to find informational and error messages from the hardware. The log is located in */var/adm/messages*.
- CCD log – the Cluster Configurations Database, which is part of the Sun Cluster configuration, logs messages here. Use this log to find informational and error messages about the Sun Cluster configuration and health. The log is located in */var/opt/SUNWcluster/ccd/ccd.log*.

Glossary

active database	In a warm standby application, a database that is replicated to a standby database. See also warm standby application .
Adaptive Server	The Sybase version 11.5 and later relational database server. If you choose the RSSD option when configuring Replication Server, Adaptive Server maintains Replication Server system tables in the RSSD database.
application programming interface (API)	A predefined interface through which users or programs communicate with each other. Open Client and Open Server are examples of APIs that communicate in a client/server architecture. RCL, the Replication Command Language, is the Replication Server API.
applied function	A replicated function, associated with a function replication definition, that Replication Server delivers from a primary database to a subscribing replicate database. The function passes parameter values to a stored procedure that is executed at the replicate database. See also replicated function delivery , request function , and function replication definition .
article	A replication definition extension for tables or stored procedures that can be an element of a publication. Articles may or may not contain where clauses, which specify a subset of rows that the replicate database receives.
asynchronous procedure delivery	A method of replicating, from a source to a destination database, a stored procedure that is associated with a table replication definition.
asynchronous command	A command that a client submits where the client is not prevented from proceeding with other operations before the completion status is received. Many Replication Server commands function as asynchronous commands within the replication system.

atomic materialization	A materialization method that copies subscription data from a primary to a replicate database through the network in a single atomic operation, using a select operation with a holdlock. No changes to primary data are allowed until data transfer is complete. Replicate data may be applied either as a single transaction or in increments of ten rows per transaction, which ensures that the replicate database transaction log does not fill. Atomic materialization is the default method for the create subscription command. See also nonatomic materialization , bulk materialization and no materialization .
autocorrection	Autocorrection is a setting applied to replication definitions, using the set autocorrection command, to prevent failures caused by missing or duplicate rows in a copy of a replicated table. When autocorrection is enabled, Replication Server converts each update or insert operation into a delete followed by an insert. Autocorrection should <i>only</i> be enabled for replication definitions whose subscriptions use nonatomic materialization.
base class	A function-string class that does not inherit function strings from a parent class. See also function-string class .
bitmap subscription	A type of subscription that replicates rows based on bitmap comparisons. Create columns using the int datatype, and identify them as the rs_address datatype when you create a replication definition. When you create a subscription, compare each rs_address column to a bitmask using a bitmap comparison operator (&) in the where clause. Rows matching the subscription's bitmap are replicated.
bulk materialization	A materialization method whereby subscription data in a replicate database is initialized outside of the replication system. For example, data may be transferred from a primary database using media such as magnetic tape, diskette, CD-ROM, or optical storage disk. Bulk materialization involves a series of commands, starting with define subscription. You can use bulk materialization for subscriptions to table replication definitions or function replication definitions. See also atomic materialization , nonatomic materialization , and no materialization .
centralized database system	A database system where data is managed by a single database management system at a centralized location.
class	See error class and function-string class .
class tree	A set of function-string classes, consisting of two or more levels of derived and parent classes, that derive from the same base class. See also function-string class .

client	A program connected to a server in a client/server architecture. It may be a front-end application program executed by a user or a utility program that executes as an extension of the system.
Client/Server Interfaces (C/SI)	The Sybase interface standard for programs executing in a client/server architecture.
concurrency	The ability of multiple clients to share data or resources. Concurrency in a database management system depends upon the system protecting clients from conflicts that arise when data in use by one client is modified by another client.
connection	A connection from a Replication Server to a database. See also Data Server Interface (DSI) and logical connection .
coordinated dump	A set of database dumps or transaction dumps that is synchronized across multiple sites by distributing an <code>rs_dumpdb</code> or <code>rs_dumptran</code> function through the replication system.
database	A set of related data tables and other objects that is organized and presented to serve a specific purpose.
database generation number	Stored in both the database and the RSSD of the Replication Server that manages the database, the database generation number is the first part of the origin queue ID (<i>qid</i>) of each log record. The origin queue ID ensures that the Replication Server does not process duplicate records. During recovery operations, you may need to increment the database generation number so that Replication Server does not ignore records submitted after the database is reloaded.
database replication definition	<p>A description of a set of database objects—tables, transactions, functions, system stored procedures, and DDL—for which a subscription can be created.</p> <p>You can also create table replication definitions and function replication definitions. See also table replication definition and function replication definition.</p>
database server	A server program, such as Sybase Adaptive Server, that provides database management services to clients.
data definition language (DDL)	The set of commands in a query language, such as Transact-SQL, that describes data and their relationships in a database. DDL commands in Transact-SQL include those using the <code>create</code> , <code>drop</code> , and <code>alter</code> keywords.
data manipulation language (DML)	The set of commands in a query language, such as Transact-SQL, that operates on data. DML commands in Transact-SQL include <code>select</code> , <code>insert</code> , <code>update</code> , and <code>delete</code> .

data server	A server whose client interface conforms to the Sybase Client/Server Interfaces and provides the functionality necessary to maintain the physical representation of a replicated table in a database. Data servers are usually database servers, but they can also be any data repository with the interface and functionality Replication Server requires.
Data Server Interface (DSI)	Replication Server threads corresponding to a connection between a Replication Server and a database. DSI threads submit transactions from the DSI outbound queue to a replicate data server. They consist of a scheduler thread and one or more executor threads. The scheduler thread groups the transactions by commit order and dispatches them to the executor threads. The executor threads map functions to function strings and execute the transactions in the replicate database. DSI threads use an Open Client connection to a database. See also outbound queue and connection .
data source	A specific combination of a database management system (DBMS) product such as a relational or non-relational data server, a database residing in that DBMS, and the communications method used to access that DBMS from other parts of a replication system. See also database and data server .
decision support application	A database client application characterized by ad hoc queries, reports, and calculations and few data update transactions.
declared datatype	The datatype of the value delivered to the Replication Server from the replication agent: <ul style="list-style-type: none">• If the replication agent delivers a base Replication Server datatype, such as <code>datetime</code>, to the Replication Server, the declared datatype is the base datatype.• Otherwise, the declared datatype must be the UDD for the original datatype at the primary database.
default function string	The function string that is provided by default for the system-provided classes <code>rs_sqlserver_function_class</code> and <code>rs_default_function_class</code> and classes that inherit function strings from these classes, either directly or indirectly. See also function string .
dematerialization	The optional process, when a subscription is dropped, whereby specific rows that are not used by other subscriptions are removed from the replicate database.
derived class	A function-string class that inherits function strings from a parent class. See also function-string class and parent class .

direct route	A route used to send messages directly from a source to a destination Replication Server, with no intermediate Replication Servers. See also indirect route and route .
disk partition	See partition .
distributed database system	A database system where data is stored in multiple databases on a network. The databases may be managed by data servers of the same type (for example, Adaptive Server) or by heterogeneous data servers.
Distributor	A Replication Server thread (DIST) that helps to determine the destination of each transaction in the inbound queue.
dump marker	A message written by Adaptive Server in a database transaction log when a dump is performed. In a warm standby application, when you are initializing the standby database with data from the active database, you can specify that Replication Server use the dump marker to determine where in the transaction stream to begin applying transactions in the standby database. See also warm standby application .
Embedded Replication Server System Database (ERSSD)	The Adaptive Server Anywhere (ASA) database that stores Replication Server system tables. You can choose whether to store Replication Server system tables on the ERSSD or the Adaptive Server RSSD. See also Replication Server System Database (RSSD) .
error action	A Replication Server response to a data server error. Possible Replication Server error actions are ignore, warn, retry_log, log, retry_stop, and stop_replication. Error actions are assigned to specific data server errors.
error class	A name for a collection of data server error actions that are used with a specified database.
exceptions log	A set of three Replication Server system tables that holds information about transactions that failed on a data server. The transactions in the log must be resolved by a user or by an intelligent application. You can use the rs_helpexception stored procedure to query the exceptions log.
Failover	<p>Sybase Failover allows you to configure two version 12.0 and later Adaptive Servers as companions. If the primary companion fails, that server's devices, databases, and connections can be taken over by the secondary companion.</p> <p>For more detailed information about how Sybase Failover works in Adaptive Server, refer to <i>Using Sybase Failover in a High Availability System</i>, which is part of the Adaptive Server Enterprise documentation set.</p>

For instructions on how to enable Failover support for non-RSSD Replication Server connections to Adaptive Server, see “Configuring the Replication System to Support Sybase Failover” in Chapter 16, “Replication System Recovery,” of the *Replication Server Administration Guide*.

- fault tolerance** The ability of a system to continue to operate correctly even though one or more of its component parts is malfunctioning.
- function** A Replication Server object that represents a data server operation such as insert, delete, select, or begin transaction. Replication Server distributes such operations to other Replication Servers as functions. Each function consists of a function name and a set of data parameters. In order to execute the function in a destination database, Replication Server uses function strings to convert a function to a command or set of commands for a type of database. See also **user-defined function**, and **replicated function delivery**.
- function replication definition** A description of a replicated function used in replicated function delivery. The function replication definition, maintained by Replication Server, includes information about the parameters to be replicated and the location of the primary version of the affected data. See also **replicated function delivery**.
- function scope** The range of a function’s effect. Functions have replication definition scope or function-string class scope. A function with replication definition scope is defined for a specific replication definition, and cannot be applied to other replication definitions. A function with function-string class scope is defined once for a function-string class and is available only within that class.
- function string** A string that Replication Server uses to map a database command to a data server API. For the `rs_select` and `rs_select_with_lock` functions only, the string contains an input template, used to match function strings with the database command. For all functions, the string also contains an output template, used to format the database command for the destination data server.
- function-string class** A named collection of function strings used with a specified database connection. Function-string classes include those provided with Replication Server and those you have created. Function-string classes can share function string definitions through function-string inheritance. The three system-provided function-string classes are `rs_sqlserver_function_class`, `rs_default_function_class`, and `rs_db2_function_class`. See also **base class**, **class tree**, **derived class**, **function-string inheritance**, and **parent class**.
- function-string inheritance** The ability to share function string definitions between classes, whereby a derived class inherits function strings from a parent class. See also **derived class**, **function-string class**, and **parent class**.

function-string variable	An identifier used in a function string to represent a value that is to be substituted at run time. Variables in function strings are enclosed in question marks (?). They represent column values, function parameters, system-defined variables, or user-defined variables.
function subscription	A subscription to a function replication definition (used in applied function delivery).
generation number	See database generation number .
heterogeneous data servers	Multi-vendor data servers used together in a distributed database system.
high availability (HA)	Very low downtime. Computer systems that provide HA usually provide 99.999% availability, or roughly five minutes unscheduled downtime per year.
hibernation mode	A Replication Server state in which all DDL commands, except admin and sysadmin commands, are rejected; all routes and connections are suspended; most service threads, such as DSI and RSI, are suspended; and RSI and RepAgent (or LTM) users are logged off and not allowed to log on. Used during route upgrades, and may be turned on for a Replication Server to debug problems.
hot standby application	A database application in which the standby database can be placed into service without interrupting client applications and without losing any transactions. See also warm standby application .
ID Server	One Replication Server in a replication system is the ID Server. In addition to performing the usual Replication Server tasks, the ID Server assigns unique ID numbers to every Replication Server and database in the replication system, and maintains version information for the replication system.
inbound queue	A stable queue used to spool messages from a Replication Agent to a Replication Server.
indirect route	A route used to send messages from a source to a destination Replication Server, through one or more intermediate Replication Servers. See also direct route and route .
interfaces file	A file containing entries that define network access information for server programs in a Sybase client/server architecture. Server programs may include Adaptive Servers, SQL Servers, gateways, Replication Servers, and Replication Agents such as LTM for SQL Server. The interfaces file entries enable clients and servers to connect to each other in a network.

latency	The measure of the time it takes to distribute to a replicate database a data modification operation first applied in a primary database. The time includes Replication Agent processing, Replication Server processing, and network overhead.
local-area network (LAN)	A system of computers and devices, such as printers and terminals, connected by cabling for the purpose of sharing data and devices.
locator value	The value stored in the <code>rs_locator</code> table of the Replication Server's RSSD that identifies the latest log transaction record received and acknowledged by the Replication Server from each previous site during replication.
logical connection	A database connection that Replication Server maps to the connections for the active and standby databases in a warm standby application. See also connection and warm standby application .
login name	The name that a user or a system component such as Replication Server uses to log in to a data server, Replication Server, or Replication Agent.
Log Transfer Language (LTL)	A subset of the Replication Command Language (RCL). A Replication Agent such as RepAgent or LTM for SQL Server uses LTL commands to submit to Replication Server the information it retrieves from primary database transaction logs.
Log Transfer Manager (LTM)	The Replication Agent program for Sybase SQL Server. See also replication agent and RepAgent thread .
maintenance user	A data server login name that Replication Server uses to maintain replicate data. In most applications, maintenance user transactions are not replicated.
materialization	The process of copying data specified by a subscription from a primary database to a replicate database, thereby initializing the replicate table. Replicate data can be transferred over a network, or, for subscriptions involving large amounts of data, loaded initially from media. See also atomic materialization , bulk materialization , no materialization , and nonatomic materialization .
materialization queue	A stable queue used to spool messages related to a subscription being materialized or dematerialized.
missing row	A row missing from a replicated copy of a table but present in the primary table.

mixed-version system	<p>A replication system containing Replication Servers of different software versions that have different capabilities based on their different software versions and site versions. Mixed-version support is available only if the system version is 11.0.2 or greater.</p> <p>For example, a replication system containing Replication Servers version 11.5 or later and version 11.0.2 is a mixed-version system. A replication system containing Replication Servers of releases earlier than release 11.0.2 is not a mixed-version system, because any newer Replication Servers are restricted by the system version from using certain new features. See also site version and system version.</p>
more columns	<p>Columns in a replication definition exceeding 250, but limited to 1024. More columns are supported by Replication Server version 12.5 and later.</p>
multi-site availability (MSA)	<p>Methodology for replicating database objects—tables, functions, transactions, system stored procedures, and DDL from the primary to the replicate database. See also database replication definition.</p>
name space	<p>The scope within which an object name must be unique.</p>
nonatomic materialization	<p>A materialization method that copies subscription data from a primary to a replicate database through the network in a single operation, without a holdlock. Changes to the primary table are allowed during data transfer, which may cause temporary inconsistencies between replicate and primary databases. Data is applied in increments of ten rows per transaction, which ensures that the replicate database transaction log does not fill. Nonatomic materialization is an optional method for the create subscription command. See also autocorrection, atomic materialization, no materialization, and bulk materialization.</p>
network-based security	<p>Secure transmission of data across a network. Replication Server supports third-party security mechanisms that provide user authentication, unified login, and secure message transmission between Replication Servers.</p>
no materialization	<p>A materialization method that lets you create a subscription when the subscription data already exists at the replicate site. Use the create subscription command with the without materialization clause. You can use this method to create subscriptions to table replication definitions and function replication definitions. See also atomic materialization and bulk materialization.</p>
online transaction processing (OLTP) application	<p>A database client application characterized by frequent transactions involving data modification (inserts, deletes, and updates).</p>

Origin Queue ID (qid)	Formed by the RepAgent, the qid uniquely identifies each log record passed to the Replication Server. It includes the date and timestamp and the database generation number. See also database generation number .
orphaned row	A row in a replicated copy of a table that does not match an active subscription.
outbound queue	A stable queue used to spool messages. The DSI outbound queue spools messages to a replicate database. The RSI outbound queue spools messages to a replicate Replication Server.
parallel DSI	Configuring a database connection so that transactions are applied to a replicate data server using multiple DSI threads operating in parallel, rather than a single DSI thread. See also connection and Data Server Interface (DSI) .
parameter	An identifier representing a value that is provided when a procedure executes. Parameter names are prefixed with an @ character in function strings. When a procedure is called from a function string, Replication Server passes the parameter values, unaltered, to the data server. See also searchable parameter .
parent class	A function-string class from which a derived class inherits function strings. See also function-string class and derived class .
partition	A raw disk partition or operating system file that Replication Server uses for stable queue storage. Only use operating system files in a test environment.
physical connection	See connection .
primary data	The definitive version of a set of data in a replication system. The primary data is maintained on a data server that is known to all of the Replication Servers with subscriptions for the data.
primary database	Any database that contains data that is replicated to another database via the replication system.
primary fragment	A horizontal segment of a table that holds the primary version of a set of rows.
primary key	A set of table columns that uniquely identifies each row.
primary site	A Replication Server where a function-string class or error class is defined. See error class and function-string class .
principal user	The user who starts an application. When using network-based security, Replication Server logs in to remote servers as the principal user.
projection	A vertical slice of a table, representing a subset of the table's columns.

publication	A group of articles from the same primary database. A publication lets you collect replication definitions for related tables and/or stored procedures and then subscribe to them as a group. You collect replication definitions as articles in a publication at the source Replication Server and subscribe to them with a publication subscription at the destination Replication Server. See also article and publication subscription .
publication subscription	A subscription to a publication. See also article and publication .
published datatype	The datatype of the column after the column-level translation (and before a class-level translation, if any) at the replicate data server. The published datatype must be either a Replication Server base datatype or a UDD for the datatype in the target data server. If the published datatype is omitted from the replication definition, it defaults to the declared datatype.
query	In a database management system, a query is a request to retrieve data that meets a given set of criteria. The SQL database language includes the <code>select</code> command for queries.
quiescent	A quiescent replication system is one in which all updates have been propagated to their destinations. Some Replication Server commands or procedures require that you first quiesce the replication system.
remote procedure call (RPC)	A request to execute a procedure that resides in a remote server. The server that executes the procedure could be a Adaptive Server, a Replication Server, or a server created using Open Server. The request can originate from any of these servers or from a client application. The RPC request format is a part of the Sybase Client/Server Interfaces.
RepAgent thread	The replication agent for Adaptive Server databases. RepAgent is an Adaptive Server thread; it transfers transaction log information from the primary database to a Replication Server for distribution to other databases.
replicate database	Any database that contains data that is replicated from another database via the replication system.
replicated function delivery	A method of replicating, from a source to a destination database, a stored procedure that is associated with a function replication definition. See also applied function , request function , and function replication definition .
replicated stored procedure	An Adaptive Server stored procedure that is marked as replicated using the <code>sp_setrepproc</code> or the <code>sp_setreplicate</code> system procedure. Replicated stored procedures can be associated with function replication definitions or table replication definitions. See also replicated function delivery and asynchronous procedure delivery .

replicated table	A table that is maintained by Replication Server, in part or in whole, in databases at multiple locations. There is one primary version of the table, which is marked as replicated using the <code>sp_setreptable</code> or the <code>sp_setrepl</code> system procedure; all other versions are replicated copies.
replication agent	A program or module that transfers transaction log information representing modifications made to primary data from a database server to a Replication Server for distribution to other databases. RepAgent is the replication agent for Adaptive Server databases. LTM (Log Transfer Manager) is the replication agent for SQL Server databases.
Replication Command Language (RCL)	The commands used to manage information in Replication Server.
replication definition	<p>Usually, a description of a table for which subscriptions can be created. The replication definition, maintained by Replication Server, includes information about the columns to be replicated and the location of the primary version of the table.</p> <p>You can also create function replication definitions; sometimes the term “table replication definition” is used to distinguish between table and function replication definitions. See also function replication definition.</p>
Replication Server	The Sybase server program that maintains replicated data, typically on a LAN, and processes data transactions received from other Replication Servers on the same LAN or on a WAN.
Replication Server Interface (RSI)	A thread that logs in to a destination Replication Server and transfers commands from the RSI outbound stable queue to the destination Replication Server. There is one RSI thread for each destination Replication Server that is a recipient of commands from a primary or intermediate Replication Server. See also outbound queue and route .
Replication Server Manager (RSM)	An application for managing and monitoring a replication system and its components. Includes status and monitoring functions, diagnostics and troubleshooting functions, asynchronous notification of user-defined events, and operational control over many aspects of the system.
Replication System Administrator	The System Administrator that manages routine operations in the Replication Server.
Replication Server System Database (RSSD)	The Adaptive Server database containing a Replication Server system tables. You can choose whether to store Replication Server system tables on the RSSD or the Adaptive Server Anywhere (ASA) ERSSD. See also Embedded Replication Server System Database (ERSSD) .

Replication Server system Adaptive Server	The Adaptive Server with the database containing a Replication Server's system tables (the RSSD).
replication system	A data processing system where data is replicated in multiple databases to provide remote users with the benefits of local data access. Specifically, a replication system that is based upon Replication Server and includes other components such as Replication Agents and data servers.
replication system domain	All replication system components that use the same ID Server.
request function	A replicated function, associated with a function replication definition, that Replication Server delivers from a replicate database to a primary database. The function passes parameter values to a stored procedure that is executed at the primary database. See also replicated function delivery , request function , and function replication definition .
route	A one-way message stream from a source Replication Server to a destination Replication Server. Routes carry data modification commands (including those for RSSDs) and replicated functions or stored procedures between Replication Servers. See also direct route and indirect route .
route version	The lower of the site version numbers of the route's source and destination Replication Servers. Replication Server version 11.5 and later use the route version number to determine which data to send to the replicate site. See also site version .
row migration	The process whereby column value changes in rows in a primary version of a table cause corresponding rows in a replicate version of the table to be inserted or deleted, based on comparison with values in a subscription's where clause.
SQL Server	The Sybase relational database pre-11.5 server.
schema	The structure of the database. DDL commands and system procedures change system tables stored in the database. Supported DDL commands and system procedures can be replicated to standby databases when you use Replication Server version 11.5 or later and Adaptive Server version 11.5 or later.
searchable column	A column in a replicated table that can be specified in the where clause of a subscription or article to restrict the rows replicated at a site.
searchable parameter	A parameter in a replicated stored procedure that can be specified in the where clause of a subscription to help determine whether or not the stored procedure should be replicated. See also parameter .

secondary truncation point	See truncation point .
site	An installation consisting of, at minimum, a Replication Server, data server, and database, and possibly a Replication Agent, usually at a discrete geographic location. The components at each site are connected over a WAN to those at other sites in a replication system. See also primary site .
site version	The version number for an individual Replication Server. Once the site version has been set to a particular level, the Replication Server enables features specific to that level, and downgrades are not allowed. See also software version , route version , and system version .
software version	The version number of the software release for an individual Replication Server. See also site version and system version .
Stable Queue Manager (SQM)	A thread that manages the stable queues. There is one Stable Queue Manager (SQM) thread for each stable queue accessed by the Replication Server, whether inbound or outbound.
Stable Queue Transaction (SQT) interface	A thread that reassembles transaction commands in commit order. A Stable Queue Transaction (SQT) interface thread reads from inbound stable queues, puts transactions in commit order, then sends them to the Distributor (DIST) thread or a DSI thread, depending on which thread required the SQT ordering of the transaction.
stable queues	Store-and-forward queues where Replication Server stores messages destined for a route or database connection. Messages written into a stable queue remain there until they can be delivered to the destination Replication Server or database. Replication Server builds stable queues using its disk partitions. See also inbound queue , outbound queue , and materialization queue .
standalone mode	A special Replication Server mode used for initiating recovery operations.
standby database	In a warm standby application, a database that receives data modifications from the active database and serves as a backup of that database. See also warm standby application .
stored procedure	A collection of SQL statements and optional control-of-flow statements stored under a name in a Adaptive Server database. Stored procedures supplied with Adaptive Server are called system procedures. Some stored procedures for querying the RSSD are included with the Replication Server software.

subscription	A request for Replication Server to maintain a replicated copy of a table, or a set of rows from a table, in a replicate database at a specified location. You can also subscribe to a function replication definition, for replicating stored procedures.
subscription dematerialization	See dematerialization .
subscription materialization	See materialization .
subscription migration	See row migration .
Sybase Central	A graphical tool that provides a common interface for managing Sybase and Powersoft products. Replication Server uses Replication Server Manager as a Sybase Central plug-in. See also Replication Server Manager (RSM) .
symmetric multiprocessing (SMP)	On a multiprocessor platform, the ability of an application's threads to run in parallel. Replication Server supports SMP, which can improve server performance and efficiency.
synchronous command	A command that a client considers complete only after the completion status is received.
system function	A function that is predefined and part of the Replication Server product. Different system functions coordinate replication activities, such as <code>rs_begin</code> , or perform data manipulation operations, such as <code>rs_insert</code> , <code>rs_delete</code> , and <code>rs_update</code> .
system-provided classes	Replication Server provides the error class <code>rs_sqlserver_error_class</code> and the function-string classes <code>rs_sqlserver_function_class</code> , <code>rs_default_function_class</code> , and <code>rs_db2_function_class</code> . Function strings are generated automatically for the system-provided function-string classes and for any derived classes that inherit from these classes, directly or indirectly. See also error class and function-string class .
system version	The version number for a replication system that represents the version for which new features are enabled, for Replication Servers of release 11.0.2 or earlier, and below which no Replication Server can be downgraded or installed. For a Replication Server version 11.5, your use of certain new features requires a site version of 1150 and a system version of at least 1102. See also mixed-version system , site version , and software version .
table replication definition	See replication definition .

table subscription	A subscription to a table replication definition.
thread	A process running within Replication Server. Built upon Sybase Open Server, Replication Server has a multi-threaded architecture. Each thread performs a certain function such as managing a user session, receiving messages from a Replication Agent or another Replication Server, or applying messages to a database. See also Data Server Interface (DSI) , Distributor , and Replication Server Interface (RSI) .
transaction	A mechanism for grouping statements so that they are treated as a unit: either all statements in the group are executed or no statements in the group are executed.
Transact-SQL	The relational database language used with Adaptive Server. It is based on standard SQL (Structured Query Language), with Sybase extensions.
truncation point	<p>An Adaptive Server database that holds primary data has an active truncation point, marking the transaction log location where Adaptive Server has completed processing. This is the primary truncation point.</p> <p>The RepAgent for an Adaptive Server database maintains a secondary truncation point, marking the transaction log location separating the portion of the log successfully submitted to the Replication Server from the portion not yet submitted. The secondary truncation point ensures that each operation enters the replication system before its portion of the log is truncated.</p>
user-defined function	A function that allows you to create custom applications that use Replication Server to distribute replicated functions or asynchronous stored procedures between sites in a replication system. In replicated function delivery, a user-defined function is automatically created by Replication Server when you create a function replication definition.
variable	See function-string variable .
version	See mixed-version system , site version , software version , and system version .
warm standby application	An application that employs Replication Server to maintain a standby database for a database known as the active database. If the active database fails, Replication Server and client applications can switch to the standby database.
wide-area network (WAN)	A system of local-area networks (LANs) connected together with data communication lines.
wide columns	Columns in a replication definition containing char, varchar, binary, varbinary, unichar, univarchar, or Java inrow data that are wider than 255 bytes. Wide columns are supported by Replication Server version 12.5 and later.

- wide data** Wide data rows, limited to the size of the data page on the data server. Adaptive Server supports page sizes of 2K, 4K, 8K, and 16K. Wide data is supported by Replication Server version 12.5 and later.
- wide messages** Messages larger than 16K that span blocks. Wide messages are supported by Replication Server version 12.5 and later.

Index

Numerics

1024 columns, limit to expressions in where clause 536

A

abort switch command 511, 512
abstract plans, replication of 484
activate subscription command 369, 394, 399
 for publications 394
 publication subscription example 399
 with suspension at replicate only clause 540
 with suspension clause 540
active database 480
 managing old active after switching 513
 restarting clients 512
Adaptive Server
 described 28
 error handling 619
 login name for Replication Server access 201
add partition command 47
adding
 a server 79
 ID server domains 97
 primary keys 302
 Replication Servers to existing systems 96
 RSSD LTM 152, 710
 searchable columns 304
admin commands 511
 described 430
admin set_log_name command 702
admin who command 700
Advanced Security option 242
alarm daemon (dAlarm) 549
allow connections command 672
alter connection command 133, 172, 178, 230, 231, 241, 324
 assigning databases to function-string classes 455

 changing maintenance user password 201
 disabling password encryption 206
alter database replication definition command 414
alter function command 690
alter function replication definition command 256, 299, 335, 345
alter function string command 466
 mapping user-defined functions 692
 replacing default function string 681
alter logical connection command 520
alter replication definition command 325
alter route command 101, 114
 changing passwords 201
 disabling password encryption 206
alter table command support for warm standby 532
alter user command
 changing passwords 205
application models
 consolidated replicate 12
 redistributed corporate rollup 14
 replicated consolidated replicate 14
applied functions
 described 336
 prerequisites for implementing 332
 setting up 338
applied stored procedures
 prerequisites for implementing 681
 setting up 682
article subscriptions, creating 396
articles
 adding to publication 317
 as objects in Sybase Central 82
 definition 309
 displaying information about 316
 dropping 318
assign action command 618
asynchronous I/O daemon (dAIO) 549
asynchronous stored procedures
 adding parameters to 690
 and non-unique user-defined function name 693

Index

- applied 679
- executing 677
- request stored procedures 680
- user-defined functions 689
- asynchronous transaction replication 2
- atomic materialization
 - create subscription command for 373
 - described 351, 352
 - text and image columns 385
 - in warm standby applications 538
- autocorrection
 - bulk materialization 361
 - enabling for nonatomic materialization 354, 367
- automatic backup, ERSSD 101

B

- Back control 69
- backup, ERSSD 108
- bars
 - displaying 68
 - hiding 68
- batch commands in function strings 470
- batch configuration parameter 180, 555
- batch ltl configuration parameter 123
- batch_begin configuration parameter 180
- batch_sz LTM configuration parameter 699
- bcp utility program 358, 499, 540
- bitmap subscriptions 387
- bulk materialization
 - autocorrection 361
 - define subscription command 375
 - described 352, 355
 - for replicated functions 344
 - in warm standby applications 539
 - methods 356
 - publication subscriptions 398
 - refreshing publication subscriptions 400
 - simulating atomic materialization 359
 - simulating nonatomic materialization 361
 - stopping primary updates 357
 - taking a snapshot 357
- buttons
 - Properties 73
 - toolbar 68

C

- case, in RCL commands xxvii
- certificate authority 242
- certificates 242
- changing
 - an object's properties 72
 - database connections 177
 - existing replication system 96, 99
 - function strings 440
 - ID Server login name and password 201
 - replication definitions 294
 - Replication Server login names for the RSSD LTM 706
 - Replication Server login names for the RSSD RepAgent 200
 - routes 161
 - RSSD primary or maintenance user 199
 - searchable columns 307
 - user passwords 205
- character sets, conversion 181
- check publication command 311, 315
- check subscription command 369, 378, 395
 - after executing switch active command 539, 540
 - example 401
 - for articles 395
 - for publication subscriptions and articles 401
 - for publications 395
- CipherSuites 243
- class-level translations 322
 - existing connections 324
 - new connections 322
 - system-defined variables 325
 - with column-level translations 330
- client application
 - described 31
 - restarting after active switch 512
- Client/Server Interfaces (C/SI), client applications for 31
- closing a property sheet 73
- clusters
 - Sun 717
 - terminology 718
- column-level translations 325
 - creating 326
 - multiple replication definitions 329
- columns

- changing in replicated tables 303, 307
- deleting from primary or replicate table 307
- IDENTITY 293
- rs_address datatype 387
- specifying for replication definition 260
- command_retry command configuration parameter 180
- commands
 - configure replication server 107, 603
 - hareg 725
 - sysadmin erssd 106, 107
- concurrency control, described 48
- configuration file 96
 - LTM 702, 705
 - Replication Server 96
 - rs_subcmp program 391
- configuration parameters
 - affecting performance 551
 - ERSSD 107
 - for parallel DSI 567
 - for replication server 100
 - rs_config system table 551
 - send_standby 416
 - viewing 130
- configuration star 144
- configure connection command, setting save interval 636
- configure logical connection command 530
 - setting DSI queue save interval 530
 - setting materialization queue save interval 531
- configure replication server command 100, 101, 107, 222, 227, 233, 603
- configure route, setting save interval 634
- Connect control 69
- connect database configuration parameter 123
- connect dataserver configuration parameter 123
- connect source permission 207
- connecting
 - to a replication environment 76–78
 - to an RSM Server domain 76–78
- connecting to Replication Server, using network-based security 235, 237
- connection manager daemon (dCM) 549
- connections
 - defined 40
 - network-based security for 230
 - setting save interval 635
- consistency
 - maintaining for replicate databases 636
 - verifying for subscriptions 390
- consolidated replicate application model 12
- Contents tab 58
- context menus 68
- context-sensitive help 59
- context-sensitive menus, shortcut to 68
- controls
 - dialog box 66
 - tab 72
- conventions
 - document style xxv
 - examples xxv
 - icons xxviii
 - syntax statements xxv
- coordinated dumps
 - creating 636
 - loading primary and replicate databases 646
 - recovering databases 646
- Copy control 69
- counter names 599
- counter status options 600
- counter types 599
- counter values
 - viewing 606
- counters 597–608
 - average value 599
 - cumulative total 599
 - dSTATS daemon thread 608
 - intrusive 602
 - last value 599
 - maximum value 599
 - overview 598
 - resetting 608
 - status options 600
 - viewing information about 605
- create article command 311, 313
- create connection command 172, 174, 455
- create database replication definition command 413
- create error class 615
- create function command 689
- create function replication definition command 335, 339
- create function string class command 451, 453

Index

- create function string command 364, 464
 - create logical connection command 496
 - create object permission 207
 - create publication 311, 312
 - create replication definition command 255, 258, 325, 326
 - create route command 149
 - create subscription command
 - and nonatomic materialization 374
 - 369, 394, 396
 - and atomic materialization 373
 - example for publications 396
 - for publications 314, 394
 - create user command 204
 - adding Replication Server login name for RSSD LTM 706
 - adding Replication Server login name for RSSD RepAgent 200
 - creating
 - database connections 174
 - function replication definitions 339, 342
 - function strings 464
 - function-string classes 451
 - multiple ID Server domains 98
 - replicate tables 381
 - replication definitions 248, 256
 - Replication Server login names 204
 - subscriptions 202, 366
 - user-defined functions 689
 - creating a replication environment 78
 - creating an RSM Server domain 78
 - current_rssd_version configuration parameter 100
 - Cut control 69
- ## D
- daemons
 - alarm (dAlarm) 549
 - asynchronous I/O (dAIO) 549
 - connection manager (dCM) 549
 - counter (dSTATS) 549
 - described 544
 - miscellaneous 549
 - recovery (dREC) 549
 - subscription retry (dSUB) 549
 - version (dVERSION) 549
 - data availability
 - fault tolerance 3
 - local access 3
 - Data Definition Language (DDL) 406
 - replicating 422
 - data limits filter mode configuration parameter 123
 - data server
 - and C/SI support 16
 - error handling 614, 619
 - ID numbers 98
 - log truncation 698
 - maintenance user login names 20
 - support for heterogeneous 16, 28
 - suspending access to 177
 - data service
 - Replication Server as 725
 - start/shutdown 725
 - Database Administrator, role of 22
 - database connections
 - attributes 174
 - changing attributes of 177
 - configuring for parallel DSI 567
 - creating 174
 - displaying 194
 - dropping 192
 - information for 174
 - managing 169, 189
 - monitoring 194
 - resuming 177
 - suspending 177
 - for warm standby applications 481
 - database generation numbers
 - adjusting during database recovery 675
 - and dumps 676
 - qid 675
 - database logs
 - determining for reload 674
 - recovering messages off-line 640
 - recovering messages online 642
 - reloading 676
 - truncated primary recovery 642, 645
 - truncation 49
 - database replication definitions 413
 - as objects in Sybase Central 82
 - database replication filters 415
 - database schema, replication definitions 299

- Database Subscription Resolution Engine (DSRE) 415
- database subscriptions
 - altering 420
 - as objects in Sybase Central 83
 - dropping 420, 421
 - managing 418
 - materialization 419
 - with table and function subscriptions 421
- databases
 - active 481
 - assigning function-string classes 455
 - customizing operations 435, 473
 - failures 646
 - logical 481
 - managed by Replication Server 194
 - preparing for replication 169, 703
 - setting log recovery 672
 - standby 481
- datatype classes
 - rs_db2_dt_class 325
 - rs_informix_dt_class 325
 - rs_mssql_dt_class 325
 - rs_oracle_dt_class 325
 - rs_sqlserver_dt_class 325
- datatype definitions 327
 - for DB2 datatypes 328
 - for Informix datatypes 329
 - for Microsoft SQL Server datatypes 328
 - for Oracle datatypes 329
- datatypes
 - IDENTITY columns 293
 - rawobject and rawobject in row 275, 280
 - rs_address 292, 387
 - text and image 487
- db_packet_size configuration parameter 180, 555
- DB2 databases, function-string class 17, 436
- db2_function_class, described 446
- dbcc settrunc Transact-SQL command 642
- deadlock detection, parallel dsi 585
- debugging
 - high availability 725
- declared datatypes 326
- default function strings, restoring 469
- default partition allocation mechanism 594
- define subscription command 369, 394, 398
 - and bulk materialization 375
 - creating publication subscriptions 398
 - for publications 394
 - publications subscription example 398
 - using with replicated functions 340
- Delete control 69
- deleting
 - transactions in the exceptions log 624
- deleting an object 74
 - shortcuts to 75
- derived function-string class, described 450
- Details control 69
- dialog boxes
 - refreshing 70
 - Table Properties 72
 - tabs on 72
 - types of 71
- direct routes 144
- directory services 37, 705
- Disconnect control 69
- disconnecting
 - from a replication environment 78
 - from RSM Server 78
- disk partitions 593
- disk_affinity configuration parameter 151, 180, 556, 563
- displaying
 - assigned actions for error numbers 619
 - bars 68
 - database connections 194
 - databases with subscriptions 390
 - DSI thread status 195
 - error class information 618
 - function-related information 472
 - icons 68
 - replication definitions 299
 - RSI thread status 166
 - subscription information 390
 - transactions in the exceptions log 622
 - users of replication system 213
 - users' permissions 214
- displays
 - updating 70
- distributed data models
 - corporate rollup 9
 - custom design 9

Index

- distributed primary fragments 9
 - redistributed corporate rollup 10
- distributed database system and replication server 4
- distributed primary fragments, consolidated replicate
 - application model 14
- distributor thread (DIST) 185, 558
 - described 546
 - disabling 520
- drag-and-drop shortcuts 65
- drop article command 312, 318
- drop connection command 114, 193, 513
- drop database replication definition command 415
- drop error class 616
- drop function command 691
- drop function replication definition 346
- drop function replication definition command 335
- drop function string class command 456
- drop function string command 467
- drop logical connection command 523
- drop publication command 311, 317, 318
- drop replication definition command 256, 305
- drop route command 114, 115
- drop subscription command 113, 369, 395, 400
 - example 400
 - example of 400
 - for articles 395
 - for publications 395
 - function replication definitions 379
 - table replication definitions 379
- drop user, dropping login names 205
- drop_repldef clause 318, 319
- Drop-down hierarchy control 69
- dropping
 - database connections 192
 - databases from the ID Server 193
 - function replication definitions 346
 - function string class 456
 - function strings 467
 - logical database connections 523
 - logical databases from the ID Server 523
 - primary keys 302
 - replication definitions 305
 - Replication Server login names 205
 - Replication Servers from existing system 112, 117
 - routes 163
 - searchable columns from the searchable columns list
 - 301, 308
 - subscriptions 203
 - user-defined functions 691
- DSI threads
 - described 45, 547
 - detecting duplicate transactions 625
 - detecting losses 670
 - displaying 195
 - executor 548, 570
 - handling losses 671
 - parallel 566, 591
 - scheduler 45, 548, 570
 - for standby database 508
 - suspending to load bulk materialization data 540
- dsi_charset_convert configuration parameter 181
- dsi_check_lock_wait configuration parameter 181
- dsi_cmd_batch_size configuration parameter 181, 556
- dsi_cmd_batch_size parameter 561
- dsi_cmd_separator configuration parameter 181
- dsi_commit_check_locks_intrvl configuration parameter 181, 556, 567
- dsi_commit_check_locks_log configuration parameter 568
- dsi_commit_check_locks_max configuration parameter 568
- dsi_commit_control configuration parameter 182, 556, 568
- dsi_exec_request_sproc configuration parameter 182
- dsi_fadeout_time configuration parameter 182
- dsi_ignore_underscore configuration parameter 182
- dsi_ignore_underscore_name configuration parameter 568
- dsi_keep_triggers configuration parameter 182
- dsi_large_xact_size configuration parameter 182, 556, 568
- dsi_max_cmds_to_log configuration parameter 182
- dsi_max_text_to_log configuration parameter 183
- dsi_max_xacts_in_group configuration parameter 556
- dsi_num_large_xact_thread configuration parameter 568
- dsi_num_large_xact_threads configuration parameter 183, 556
- dsi_num_threads configuration parameter 183, 556, 568
- dsi_partitioning_rule configuration parameter 183, 557, 568

dsi_replication configuration parameter 183
 dsi_serialization_method configuration parameter
 184, 557, 569
 dsi_sql_data_style configuration parameter 184
 dsi_sqt_max_cache_size configuration parameter
 185, 557
 dsi_text_convert_multiplier configuration parameters
 185
 dsi_text_max_xacts_in_group configuration parameter
 182
 dsi_xact_group_size configuration parameter 185,
 557
 dSTATS daemon 549, 608
 dump command 358
 dump database command 504, 637
 dump marker option for rs_init program 501, 514
 dump transaction command 504, 637
 dump_load configuration parameter 185
 dumps
 creating 636
 database generation numbers 676
 determining for reload 674
 initializing warm standby databases 499, 504
 transaction timestamp 674

E

edges
 topology view 84, 85
 edit menu 67
 empty function strings, creating 470
 enable replication marker 499
 enabling flushing 602
 enabling RepAgent 121
 enabling sampling 601
 encrypted passwords
 sending 202
 encryption
 disabling for Replication Server 206, 707
 enabling for Replication Server 206, 707
 error classes
 changing primary Replication Server 617
 creating 615
 dropping 616
 initializing 616

Open Server gateway 17
 rs_sqlserver_error_class 615
 error handling
 assigning actions 618
 data server 615, 619
 general 609
 LTM 712
 Replication Server 610, 712
 system transactions 626
 error log files
 beginning a new Replication Server log file 613
 described 610
 displaying current log file name 613
 log file for LTM 701
 LTM 613, 701
 Replication Server 426, 610
 error messages
 format 611
 Replication Server login name 429
 severity levels 611
 system transactions 626
 errors
 log file for Replication Server 426
 standard error output 426, 702
 errsd_backup_dir 107
 errsd_backup_interval 107
 errsd_backup_start_time 107
 ERSSD
 automatic backup 101
 created by rs_init program 97
 media failure, recovery from 110
 recovery instructions 109
 recovery procedures 109
 use isql to execute 92
 ERSSD (Embedded Replication Server System
 Database) 105
 ERSSD (Embedded Replication Server System
 Database), configuring 106
 ERSSD backup directory path 106
 ERSSD configuration parameters 107
 ERSSD database file path 106
 ERSSD transaction log file path 106
 ERSSD transaction log mirror file path 106
 ERSSD, backup directory files 108
 ERSSD, configuration parameters in rs_init table 100
 ERSSD, files, moving 107

Index

- ERSSD, users 108
- erssd_backup_start_date 107
- establishing a replication environment 78
- establishing an RSM Server domain 78
- examples
 - assigning domain ID numbers 98
 - atomic materialization 353
 - DSI loss detection 670
 - replication definition 258
 - routing 151
 - rs_subcmp configuration file 391
 - SQM loss detection 669
 - style conventions xxv
 - warm standby application 507
- exceptions log
 - accessing 622
 - deleting transactions 624
 - displaying transactions 622
 - exceptions handling 620
 - transactions written to 49
- exec_cmds_per_timeslice configuration parameter 185, 551, 557, 562
- exec_sqm_write_request_limit configuration parameter 185, 551, 558
- exec_sqm_write_request_limit parameter 562
- executing
 - RCL commands 91
 - scripts with isql 93
- extended limits 129, 269, 275
 - more columns 270
 - wide columns 270
 - wide data 270
 - wide messages 271
- F**
- failed transactions
 - handling 620, 624
 - process for resolving 621
- failover, support for in Replication Server 628
- failure
 - data server 609
 - network 609
- fault tolerance, achieving 3
- file menu 67
- files
 - interfaces 38, 79, 704
 - LTM configuration 705
 - LTM error log 701
 - moving, ERSSD 107
 - Replication Server configuration 96
 - Replication Server error log 426
 - Replication Server run file 94
 - RSM.Servers 202
 - sql.ini 79
 - standard error output 426, 702
- find tab 59
- finding current save interval 633
- flushed values
 - viewing 607
- flushing
 - for connection parameters 604
 - for route parameters 604
 - multithreaded modules 604
 - resetting after 602
 - setting intervals for 603
- folder icons 62
- folders
 - topology 85
- for new articles clause 397
- Formatting, RCL commands xxvii
- function replication definitions
 - commands for managing 335
 - sending parameters to standby database 536
 - subscribing to 344
- function scope, described 439
- function strings
 - changing 440
 - changing replication definitions 300
 - creating 464
 - creating empty 470
 - customizing 423
 - defined 42
 - defining multiple commands 470
 - described 442
 - dropping 467
 - examples 465
 - for Java columns 281
 - generated for standby databases 484
 - input templates 457
 - managing 457, 471

- none 476
- output templates 457
- remapping table and column names 470
- restoring default 469
- restoring defaults with output template 469
- updating 466
- variables 42, 462
- writetext 476
- functions
 - described 437
- function-string classes 323
 - assigning to databases 455
 - changing the primary Replication Server 454
 - creating 451
 - defined 43
 - described 445
 - dropping 456
 - for Adaptive Server databases 17
 - for DB2 databases 17, 436
 - managing 450, 454
 - open architecture 17
 - rs_default_function_class 484
 - rs_informix_function_class 323
 - rs_mssql_function_class 323
 - rs_oracle_function_class 323
 - rs_sqlserver_function_class 323
- function-string inheritance 450

G

- grant command 171, 207, 212, 506
- graphic icons xxviii

H

- ha failover configuration parameter, RepAgent 123
- ha_failover configuration parameter 103, 105, 631
- hareg command 725
- HDS. See heterogeneous datatype support
- help
 - context-sensitive 59
 - for Replication Server plug-in 57–59
 - menu 68
 - search 59

- toolbar buttons 69
- topic 57
- help contents 58
- help index 59
- heterogeneous data servers 423
 - as objects in Sybase Central 82
- heterogeneous datatype support 319–330
 - class-level translations 322
 - column-level translations 325
 - data servers 319
 - function-string classes for 323
 - overview 320
 - procedure for 321
- hiding
 - bars 68
 - icons 68
- high availability 717–726
 - configuring Replication Server for 720
 - configuring Sun Cluster for 720
 - installing Replication Server for 721
 - scripts 719
 - technology overview 719
 - terminology 718
- hints 595

I

- icons xxviii
 - displaying 68
 - folder 62
 - hiding 68
 - object 62
- ID numbers
 - data servers 98
 - Replication Server 98
- ID Server 96
 - adding server domains 97
 - dropping a database from 193
 - dropping a logical database from 523
 - guidelines 98
 - ID numbers 99
 - login name 27
 - login name and password 200
 - network-based security for 233
 - requirements 27

Index

- specifying domain ID numbers 98
- id_msg_confidentiality parameter 233
- id_security_mechanism parameter 233
- id_server configuration parameter 100
- identifiers
 - case sensitivity xxvii
- IDENTITY columns 293
- ignore loss command
 - handling losses 671
 - ignoring SQM and DSI losses 672
 - ignoring SQM loss after setting log recovery 673
 - and warm standby applications 542
- image datatype
 - changing replication for 288
 - LTM shutdown and 289
 - overview of replication 283
- inbound queue
 - defined 44
 - displaying reader threads 516
 - multiple reader threads 521
- inconsistencies
 - correcting 391
 - locating 391
 - occurring in tables 390
 - resulting from skip transaction clause 189
- index tab 59
- indirect routes 145
- informational messages
 - format 611
- init_sqm_write_delay configuration parameter 552
- init_sqm_write_max_delay configuration parameter 552
- input templates, example 461
- installing Replication Server
 - as a data service 722
 - for HA 721
- interfaces file 38, 79, 704
 - checking for accuracy 428
 - defined 38
 - modifying for warm standby application 517
 - requirements 38, 704
- intrusive counters 602
 - TranTimeAve 602
 - TranTimeLast 602
 - TranTimeMax 602
- isql interactive SQL utility 92, 202
 - executing RCL commands 91

- executing scripts 93
- to execute ERSSD 92
- verifying server status 429

J

- Java columns
 - function strings 281
- Java datatypes 280

K

- keyboard shortcuts 66
- keytab file 222, 235

L

- language
 - function string output templates 459
- Large Icon control 69
- large messages 274
- large transactions 570
- LDAP server 39
 - Open Client/Server 40
- libtcl.cfg file 217
- List control 69
- load database command 504
- load transaction command 504
- loading
 - primary database from dumps 647
- log recovery
 - detecting losses 673
 - setting for databases 672
- log transfer
 - resuming 133
 - suspending 132, 133
- Log Transfer Language (LTL) 29
- log truncation, Adaptive Server 49
- logical connection
 - configuring materialization queue save interval 530
 - configuring save interval 530
 - creating 495

- send standby_repdef_cols configuration parameter 520
 - logical database connections
 - dropping 523
 - logical views 81
 - login names
 - for subscriptions 367
 - applied functions 203
 - applied stored procedures 203
 - creating for maintenance user 172
 - creating for Replication Server 204
 - data server 20
 - dependencies 198, 203
 - displaying maintenance user 172
 - dropping Replication Server 205
 - for SQL Server use 707
 - ID Server 27, 200
 - list of commands for managing 204
 - for LTM users 706
 - Replication Server 20
 - Replication Server for RSSD LTM use 706
 - Replication Server for RSSD RepAgent use 200
 - for Replication Server use 201
 - request functions 203
 - request stored procedures 203
 - RSM.Servers file 202
 - RSSD maintenance user 199
 - RSSD primary user 199
 - for subscriptions 203
 - loss detection
 - after setting log recovery 673
 - checking messages 670
 - DSI loss 668, 670
 - handling losses 671
 - preventing false losses in stable queue 669
 - rebuilding stable queues 668
 - SQM loss 668
 - with warm standby applications 542
 - LTM
 - adding for RSSD 710
 - checking for errors 701
 - configuration file 705
 - data flow 697
 - described 29, 696
 - error handling 712
 - error log file 712
 - errors from SQL Server 713
 - login name and password 706
 - login names to access SQL Server 707
 - normalization errors and 714
 - origin queue ID (qid) 675
 - replicate databases 702
 - requirements 702
 - resuming 709
 - for RSSD 706
 - shutting down 701
 - starting 701
 - suspending 708
 - transaction timestamp 674
 - truncation point 698
 - LTM command 701
 - LTM truncation point
 - described 698, 699
- ## M
- maintenance user
 - changing passwords 201
 - described 201
 - displaying list of 172
 - granting database access 172
 - login names 20, 172
 - required permissions 171, 172
 - RSSD 199
 - for standby database 506
 - managing replication systems
 - with Sybase Central 75
 - mapping security-system login 240
 - marking data for replication 412
 - master database
 - and warm standby applications 483
 - materialization
 - database subscriptions 419
 - MSA 419
 - materialization methods
 - for function replication definitions 397
 - for publication subscriptions 397
 - materialization queue save interval
 - setting for logical connections 530
 - strict setting 530
 - materialization_save_interval configuration parameter

Index

- for logical connections 519
- md_source_memory_pool configuration parameter 186
- md_sqm_write_request_limit configuration parameter 185, 552, 558
- md_sqm_write_request_limit parameter 562
- media failure, ERSSD, recovery 110
- memory_limit configuration parameter 552
- memory_max configuration parameter 187
- menus
 - shortcut 68
 - Sybase Central 67
 - Tools 67
 - View 67
- menus and toolbars 66
- Message Delivery module (MD) 547
- messages
 - handling loss in stable queues 671
 - recovering from off-line database logs 640
 - recovering from online database logs 642
 - SQM loss detection 673
- minimal columns
 - specifying for replication 257, 265
- minimum_rssd_version configuration parameter 100
- mixed versions
 - replication system 18
- mnemonics 66
- modifiers
 - in function string variables 462
- modules 598
 - described 544
 - Message Delivery 547
 - multi-threaded 598
 - non-threaded 599
 - overview 598
 - single-threaded 599
 - Transaction Delivery 547
- monitoring
 - database connections 194
 - errors 426, 702
 - partition percentages 433
 - Replication Server 428
 - routes 165
- more columns 270
- mount command 357, 499
- move primary command 113, 454, 617
 - routing requirements 143
- moving ERSSD files 107
- msg_confidentiality parameter 226
- msg_integrity configuration parameter 226
- msg_origin_check configuration parameter 226
- msg_replay_detection configuration parameter 226
- msg_sequence_check configuration parameter 226
- multiple replication definitions
 - and function strings 444
 - column-level translations 329
- multiprocessor platforms 592
- multiprocessors
 - enabling 593
 - monitoring 593
- multi-site availability (MSA) 33, 350, 405
 - advantages of 406
 - bulk materialization 408
 - concurrent replication definitions 416
 - data replication 412
 - database replication definitions 413
 - dropping database replication definitions 415
 - features of 405
 - function strings 423
 - heterogeneous data servers 423
 - marking data 412
 - replicating tables and functions separately 408
 - replicating the database 407
 - resynchronizing tables 414
 - set up 407
 - warm standby 409
- mutual_auth 226

N

- name space, replication definition 258
- naming replicated tables 306
- network-based security 214–242, 705
 - activating 222
 - altering 238
 - configuring services 223
 - credential 214
 - disabling 238
 - environment variables for 220
 - global settings 228
 - how it works 215
 - logging in 235

- mapping login 240
- message protection 215
- parameters 226
- pathways 224
- planning for 227
- potential security breach 241
- requirements 216
- requirements and restrictions 216
- restrictions 216
- setting up 217
- using multiple security mechanisms 241
- viewing information about 240
- Next control 69
- no materialization method
 - described 351
 - describing 355
 - requirements for using 355
- nonatomic materialization
 - autocorrection 367
 - described 351, 353
 - text and image columns 385
 - in warm standby applications 539
- none
 - transaction serialization method 574
- none function string output templates 476
- normalization errors
 - LTM and 714
- num_threads configuration parameter 188

O

- object icons 62
- object properties 71, 73
- object property sheets 71
- object tree 61, 63
 - moving through 62
- objectid.dat file 218, 219
- objects
 - articles 82
 - creating in Sybase Central 70
 - database replication definitions 82
 - database subscriptions 83
 - deleting 74
 - heterogeneous data servers 82
 - publication subscriptions 83

- publications 82
 - selecting 62
- stored procedure subscriptions 83
- stored procedures 83
- Sybase Replication Agents 82
- table replication definitions 82
- table subscriptions 83
- tables 82
- online database command 504
- online help 57, 57–59
- Open Client Client-Library 91
- Open Server gateway
 - creating for Replication Server 16
- OQID commit stack 582
- origin queue ID (qid) 674
 - determining database generation numbers 675
- oserver configuration parameter 100
- outbound queue, defined 45
- output templates
 - creating empty function strings 470
 - language 459
 - none 476
 - restoring default function strings 469
 - rpc 459
 - writetext 476

P

- parallel DSI
 - benefits and risks 566
 - components for 570
 - conflicting updates 590
 - deadlocks 586
 - described 566, 591
 - function strings for 585, 586
 - grouping logic 580
 - infrequent conflicting updates 590
 - optimal performance 587
 - OQID commit stack 582
 - parameters for 567
 - partitioning rules 576, 589
 - reducing contentions 588
 - resolving conflicts 581
 - serialization methods 572
 - setting parameters for 569

Index

- transaction size 590
 - parallel_dsi configuration parameter 558, 569
 - parameters
 - disk_affinity 563
 - dsi_cmd_batch_size 561
 - exec_cmds_per_timeslice 562
 - exec_sqm_write_request_limit 562
 - RepAgent configuration 97, 123
 - stats_daemon_sleep_time 603
 - stats_flush_rssd 602
 - stats_reset_afterflush 602
 - stats_sampling 601
 - parameters, stored procedure
 - adding to replicated functions 345
 - adding to user-defined functions 690
 - parent function-string class 450
 - partition affinity
 - allocation hint 595
 - alter connection command 595
 - alter route command 595
 - default allocation 594
 - rs_diskaffinity system table 595
 - partition failure
 - recovering 637, 642
 - partitioning rules 576, 589
 - none 577
 - origin begin and commit times 578
 - transaction name 579
 - user name 577
 - partitions 593
 - guidelines for choosing 47
 - monitoring percentages 433
 - recovering from loss or failure 637, 642
 - space requirements 635
 - password encryption
 - replication system 21
 - password_encryption configuration parameter 206
 - password_encryption LTM configuration parameter 707
 - passwords
 - alter user command 205
 - applied functions 203
 - applied stored procedures 203
 - changing 205
 - changing for maintenance user 201
 - changing for Replication Server in RSSD LTM 706
 - changing for Replication Server in RSSD RepAgent 200
 - changing for RSSD primary user 199
 - dependencies 201, 203
 - enabling encryption 206, 707
 - encrypted 207
 - for LTM use 706
 - for SQL Server use 707
 - ID Server 200, 201
 - for LTM use 706
 - for RepAgent use 200
 - for Replication Server use 201
 - request functions 203
 - request stored procedures 203
 - requirements for Replication Server 204
 - RSM.Servers file 202
 - subscriptions 203
- Paste control 69
 - performance
 - replicating local data 3
 - routing 147
 - permissions
 - creating subscriptions 366
 - displaying for users 214
 - dropping subscriptions 379
 - granting 211
 - granting database access for maintenance user 172
 - maintenance user 172
 - managing for Replication Server 207, 214
 - revoking 212
 - subscription requirements 367
 - summary of commands for 209
 - system for 21
 - physical views 81, 83
 - prev_min_rssd_version configuration parameter 100
 - prev_rssd_version configuration parameter 100
 - primary data 48
 - failure to update 48
 - primary data server
 - subscription requirements 367
 - primary databases
 - loading from dumps 647
 - LTM configuration file 705
 - recovering from failure 646
 - recovering truncated logs 642, 645
 - required permissions 173
 - subscription requirements 367

- upgrading from replicate only 710
- primary dumps
 - recovering primary databases 646
- primary key
 - adding or dropping 302
 - defined 32, 257
 - requirement for unique 250
 - for tables in a warm standby database 535
- primary key clause 263
- primary key columns
 - restrictions on updating 250
- primary Replication Server
 - changing for an error class 617
 - changing function-string class to another Replication Server 454
 - processing in 544, 549
- primary subscribe permission 207
- primary tables
 - subscription requirements 366
 - upgrading Replication Server to manage 710
- primary user
 - RSSD 199
- principal user 222
- priority configuration parameter 123
- procedure
 - enabling sampling and flushing 601
- Properties
 - button 73
 - control 69
- properties
 - object 71, 73
- property sheets 71
 - Table Properties 72
 - tabs on 72
- publication subscriptions 393, 402
 - activating 398
 - as objects in Sybase Central 83
 - bulk materialization method 398
 - creating 396
 - defining 398
 - definition 309
 - dropping 400
 - monitoring 401
 - refreshing 397, 399
 - restrictions 393
 - specifying materialization methods 397

- status information 401
 - validating 399
- publications 319
 - adding articles 317
 - altering 316
 - as objects in Sybase Central 82
 - creating at the command line 312
 - definition 309
 - displaying information about 315
 - dropping 317
 - dropping replication definitions 318
 - for stored procedures 347
 - from Sybase Central 310
 - isql script 310
 - procedure for creating 308
 - procedure for Sybase Central 310
 - RCL commands for 311
 - viewing information about 315
- published datatype 326

Q

- queries
 - for exceptions log system tables 624
- queue ID 674
- quiesce database ... to manifest_file command 499
- quiescing
 - procedure for Replication Server 111
 - replication system 111

R

- RCL commands 689
 - abort switch command 511, 512
 - activate subscription command 369
 - admin log_name command 613
 - admin logical_status command 511, 515
 - admin set_log_name 613
 - admin set_log_name command 426
 - admin who, sqm command 633
 - allow connections command 672
 - alter connection command 172, 178, 455
 - alter function command 690
 - alter function replication definition command 335

- alter function string command 347, 466
- alter replication definition command 256, 300
- assign action command 618
- check subscription command 369, 378
- configure connection command 471, 522, 637
- create connection command 174, 455
- create error class command 615
- create function replication definition command 335, 339, 342
- create function string class command 453
- create function string command 465
- create logical connection command 496
- create replication definition command 255, 258
- create route command 149
- create subscription command 369, 373, 374
- create user command 706
- define subscription command 369, 375
- drop connection command 193, 513
- drop error class command 616
- drop function replication definition command 335, 346
- drop function string class command 456
- drop function string command 467
- drop replication definition command 256
- drop route command 163
- drop subscription command 369, 379
- drop user command 205
- executing command 91
- formatting xxvii
- grant command 212
- ignore loss command 671, 674
- move primary command 454, 617
- rebuild queues command 665
- resume connection 505
- resume connection command 188, 505, 621
- resume log transfer command 709
- resume route command 155
- revoke command 212
- set autocorrection command 367
- set log recovery command 672
- shutdown command 96
- suspend connection command 178, 621
- suspend log transfer command 132, 708
- suspend route 154
- sysadmin dropdb command 193
- sysadmin dropldb command 524
- sysadmin purge_route_at_replicate command 164
- sysadmin restore_dsi_saved_segments command 635
- table of permissions 209, 211
- validate subscription command 369
- wait for create standby command 505
- wait for switch command 511
- rebuild queues command 665
- rec_daemon_sleep_time configuration parameter 552, 560
- recovery
 - from primary database failures 646
 - of messages from off-line database logs 640
 - overview 650
 - partition loss or failure 637, 642
 - from RSSD failure 650, 664
 - of RSSD from dumps 651
 - setting save intervals 633
 - support tasks 664, 676
 - from truncated primary database logs 642, 645
 - using procedures 628
- recovery daemon (dREC) 549
- recovery instructions, ERSSD 109
- recovery mode
 - Replication Server 648, 666, 672
- recovery procedures, ERSSD 109
- recovery, from media failure, ERSSD 110
- redistributed corporate rollup application model 14
- refreshing windows and dialog boxes 70
- rep_as_standby configuration parameter 186
- RepAgent 119
 - configuration parameters 122, 123
 - described 29
 - disabling 127
 - enabling for databases 121
 - enabling on Adaptive Server 121
 - error log messages 613
 - error messages 127
 - extended limits 129
 - network security for 128
 - network-based security for 234
 - parameters 97
 - for RSSD 200
 - secondary truncation point 49
 - setting up 120
 - starting 125

- starting up 125
- status information 129
- stopping 126
- suspending 132
- thread status 131
- thread user status 131
- truncation point 49
- upgrading from LTM 120
- RepAgent Executor 185, 557
- RepAgent user thread 545
- replicate databases
 - preventing data loss 633
 - upgrading to primary databases 189, 710
- replicate minimal columns
 - and non-default function strings 475
 - and rs_default_fs system variable 475
- replicate Replication Server
 - processing in 549
 - subscription requirements 367
- replicate tables
 - requirements for subscriptions 366
- replicated consolidated replicate application model 14
- replicated functions
 - adding parameters 345
 - adding searchable parameters 345
 - described 34, 336
 - dropping 345
 - modifying 345
 - subscribing to 340
- replicated stored procedures
 - enabling for replication 344, 688
 - login and password dependencies 203
 - replicated function delivery 336
- replicated tables
 - changing 305
 - changing searchable columns 307
 - commands for modification 255
 - dropping 306
 - enabling for replication 276
 - failed updates 48
 - procedures for changing 306
 - renaming primary and replicate copies 306
 - requirements 32
 - subscribing to 350
- replication
 - configuring in standby databases 522
- Replication Agent
 - described 28
 - open architecture 16
 - requirements 31
- replication definitions
 - for distributed primary fragments 12
 - changing 294
 - commands for managing 255
 - creating 256
 - datatypes 260
 - defined 33
 - described 257
 - dropping 305
 - dropping from articles 318
 - dropping from publications 318
 - examples 258
 - extended limits 269
 - functions 336
 - name space 258
 - for distributed primary fragments 9
 - primary key 263
 - required for warm standby 531
 - requirements for creating subscriptions 366
 - rs_address datatype 387
 - searchable columns 264
 - sending columns to standby database 536
 - text and image columns 267
 - text or image columns 284
 - using 256, 305
 - using for tables with more than 1024 columns 536
- replication environment 55
 - connecting to 76–78
 - defining 78
 - disconnecting from 78
- Replication Manager 53, 54, 55
 - tasks 76
- Replication Server
 - adding to an existing system 96, 703
 - advantages of 3
 - checking for errors 426, 701
 - configuration file 96, 229
 - configuring rs_config system table 99
 - connections 37
 - described 27
 - distributed data models 9

- dropping from existing system 112, 117
- error log 514, 610
- executable program 95
- general description 1
- handling lost messages 671
- and heterogeneous data servers 16
- ID numbers 98
- informational messages 611
- internals 543, 550
- introduction 1
- list of databases managed by 194
- log recovery mode 672
- login name for Adaptive Server use 201
- login name for LTM use 706
- login name for RSSD use 201
- managing 87, 112
- managing login names 203
- monitoring 428
- partitions 432, 433
- permissions 207, 212
- plug-in to Sybase Central 53
- primary copy model 6
- processing in primary 544, 549
- processing in replicate 549
- quiescing 111
- rebuilding stable queues 665
- recovery mode 648, 666, 672
- role in a distributed database system 4
- run file for 94
- security 214
- shutting down 96
- standalone mode 640, 665
- standard errors 426, 702
- starting 94
- subscription requirements 367
- system data flow 7, 696
- technical overview 25, 47
- transaction handling 43
- upgrading to manage primary tables 710
- verifying a working system 426
- verifying status 429
- Replication Server plug-in 54
 - help 57–59
 - tasks 76
- Replication Server programs
 - ltm 701
 - repsrver 95
 - rs_init 96, 703
 - rs_subcmp 391, 671
- Replication Server System Database (RSSD)
 - described 30
 - login names 199
 - LTM for 702
 - maintaining 99
 - managing 103
 - recovering from failure 650
 - RepAgent for 200
 - requirements 31
 - rs_helpdb stored procedure 194
 - rs_maintusers 213
 - rs_users 213
 - system tables 30
 - updating database generation numbers 676
 - users 199
- replication system
 - components 25, 32
 - creating multiple domains 98
 - domains 97
 - error log files 610
 - open architecture 16
 - preventing data loss 633
 - quiescing 111
 - roles and responsibilities 22
 - security 197
 - setting up 87
- Replication System Administrator
 - role of xx, 22
- repsrver command 95
- request functions
 - defined 35
 - described 337
 - login names and passwords 203
 - permissions needed at primary 173
 - prerequisites for implementing 332
 - setting up 341
- request stored procedures 680
 - login names and passwords 203
 - prerequisites for implementing 681
 - primary copy model 6
 - setting up 686
- requirements
 - Replication Server 25, 702

- restoring
 - dumps 636
 - primary and replicate databases 646
 - RSSD 651
- restrictions
 - on replicated data 249
 - warm standby applications 483
- resume connection command 188, 231, 362, 505, 621
- resume log transfer command 132, 709
- resume route command 155
- resuming
 - log transfer 133
 - RepAgent 133
 - routes 154
- resynchronizing replicate tables 414
- retry timeout configuration parameter 124
- revoke command 207, 212
- RMP tasks 76
- roll-up
 - consolidated replicate application model 12
 - consolidated replicate as primary application model 14
- route version
 - between Replication Servers 164
- routes
 - changing 155, 161
 - creating 148
 - creating login names 150
 - defined 40
 - determining 142
 - direct 144
 - dropping 163
 - indirect 145
 - managing 141, 162
 - monitoring creation of 165
 - network-based security for 231
 - purging 116
 - requirements 142
 - resuming 154
 - RSSD recovery 664
 - setting save interval 634
 - subscriptions 147
 - suspending 154
 - unsupported 148
 - upgrading 164
- routing
 - examples 152
 - overlapping subscriptions 147
 - schemes 144
- row migration
 - text and image columns 386
- RPC function string output templates 459
- rs name configuration parameter 123
- rs password configuration parameter 123
- RS user thread 549
- rs username configuration parameter 123
- rs_address datatype 292
- rs_begin system function 440
- rs_check_repl system function 440
- rs_commit system function 440
- rs_config system table 101
 - configuration parameters 551
- rs_databases system table 194
- rs_datarow_for_writetext system function 442
- rs_default_function_class 484
 - described 446
- rs_delete system function 442
- rs_delexception stored procedure 624
- rs_diskaffinity system table 595
- rs_dumpdb system function 440, 637
- rs_dumptran system function 440, 637
- rs_exceptscmd system table 622
- rs_exceptshdr system table 622
- rs_get_charset system function 441
- rs_get_lastcommit system function 441
- rs_get_sortorder system function 441
- rs_get_textptr system function 442
- rs_get_thread_seq system function 441, 586
- rs_get_thread_seq_noholdlock system function 441, 586
- rs_helpclass stored procedure 473
- rs_helpdb stored procedure 113, 194
- rs_helperror stored procedure 619
- rs_helpexception stored procedure 622
- rs_helpfstring stored procedure 473
- rs_helpfunc stored procedure 473
- rs_helppub stored procedure 312, 316, 402
- rs_helpprepdb stored procedure 390
- rs_helproute stored procedure 166
- rs_helpuser stored procedure 213
- rs_idnames system table

Index

- dropping database from 193
 - dropping logical database from 524
 - rs_init
 - creates ERSSD 97
 - rs_init program 100
 - adding a standby database 504
 - adding Replication Servers to existing system 703
 - adding warm standby databases 496
 - enabling password encryption 707
 - preparing databases for replication 703
 - rs_init_erroractions stored procedure 616
 - rs_initialize_threads system function 441, 586
 - rs_insert system function 442
 - rs_lastcommit system table 188
 - permissions 172
 - rs_maintusers system table 213
 - rs_marker stored procedure 173
 - rs_marker system function 441
 - rs_mk_rsids_consistent stored procedure 656
 - rs_raw_object_serialization function 441
 - rs_repl_off system function 441
 - rs_rollback system function 441
 - rs_select system function 442
 - updating function strings 466
 - rs_select_with_lock system function 442
 - updating function strings 466
 - rs_set_isolation_level3 system function 441, 575
 - rs_set_proxy function 441
 - rs_setproxy function string 238
 - rs_sqlserver_error_class error class 615
 - rs_sqlserver_function_class 454, 470
 - described 445
 - rs_statcounters system table 598, 605
 - rs_statdetail system table 598
 - rs_statrun system table 598
 - rs_subcmp program 391, 541, 671
 - rs_subscriptions system table 398
 - rs_systext system table 622
 - rs_textptr_init system function 442
 - rs_thread_check_lock system function 441
 - rs_triggers_reset system function 441
 - rs_trunc_reset system function 441
 - rs_trunc_set system function 441
 - rs_truncate function 442
 - rs_update system function 442
 - rs_update_lastcommit stored procedure 173
 - rs_update_threads system function 441, 586
 - rs_usedb system function 441
 - rs_users system table 213
 - rs_writetext system function 442
 - RSI threads
 - described 46, 548
 - displaying 166
 - RSI user thread 550
 - rsi_batch_size configuration parameter 151, 558
 - rsi_fadeout_time configuration parameter 151
 - rsi_packet_size configuration parameter 151, 558
 - rsi_sync_interval configuration parameter 151, 558
 - rsi_xact_with_large_msg configuration parameter 151
 - RSM
 - login name and password dependencies 202
 - monitoring errors 426
 - tasks 76
 - verifying server status 429
 - RSM Server 55
 - connecting to 76–78
 - disconnecting from 78
 - RSM Server domain 55, 63
 - defining 78
 - RSSD
 - network-based security for 229
 - RSSD failure
 - recovering 650, 664
 - rssd_error_class configuration parameter 100
 - run file, Replication Server 94
- ## S
- sa permission xx, 207, 209
 - sampling
 - of non-intrusive counters 601
 - save interval
 - described 633
 - setting for connections 636
 - setting for logical connections 530
 - setting for routes 634
 - strict setting 530, 539
 - save_interval configuration parameter 633
 - for database connection 186
 - for logical connection 520
 - for route 151

- scan batch size configuration parameter 124
- scan timeout configuration parameter 124
- scan_retry LTM configuration parameter 699
- schema cache growth factor configuration parameter 124
- scope, of functions 439
- scripts
 - executing in isql 93
 - replication definition examples 381
 - verifying server status 429
- search
 - help 59
- searchable columns
 - adding searchable columns 304
 - dropping from the searchable columns list 301, 308
- searchable columns clause 257
- searchable parameters
 - adding to replicated functions 345
- secondary truncation point
 - and disabling RepAgent 127
 - described 49, 50
- secure sockets layer 242
- security
 - network-based 214
 - RepAgent 128
 - Replication Server 197, 214
 - replication system 20
- security mechanisms
 - CyberSafe Kerberos 216
 - DCE 218
 - Transarc DCE 216
- security services, configuring 223
- security, network-based 204–242
- security_mechanism 226
- select command 358
- select with holdlock clause 397
- send buffer_size configuration parameter 124
- send maint xacts to replicate configuration parameter 124
- send standby clause
 - for columns 536
 - for parameters 536
- send standby_repdef_cols configuration parameter for logical connections 520
- send structured opids configuration parameter 124
- send warm standby xacts configuration parameter 124
- send_enc_password configuration parameter 202, 552
- send_standby configuration parameter 416
- serialization methods
 - isolation_level_3 574
 - no_wait 573
 - none 573
 - single_transaction_per_origin 575
 - wait_for_commit 574
 - wait_for_start 573
- server user's ID
 - for warm standby databases 503
- servers
 - adding 79
 - verifying operation 429
- set autocorrection command 367
- set function string class clause 455
- set log recovery command 672
- set proxy command 221, 238
- set replication Transact-SQL command 493, 522
- set triggers off Transact-SQL command 522
- setting up network-based security 217
- severity levels
 - data server errors 619
 - error messages 611
 - LTM 712
 - Replication Server 619
- short ltl keywords configuration parameter 125
- shortcut menus 68
- shortcuts
 - to create a new object 71
 - to delete an object 75
 - to view an object's properties 72
- shutdown RCL command 96
- shutting down
 - LTM 701
 - Replication Server 96
- single_transaction_per_origin
 - transaction serialization method 575
- skip ltl errors configuration parameter 125
- skip transaction clause 188, 622
- skip unsupported features configuration parameter 125
- Small Icon control 69
- small transactions 570
- smp_enable configuration parameter 553

Index

- sp_config_rep_agent system procedure 234
- sp_helpcounter command system procedure 605
- sp_reptostandby system procedure 487, 505
- sp_role system procedure 171
- sp_setreplicate system procedure
 - marking rs_marker for replication 191
 - marking stored procedures for replication 688
- sp_setrepproc system procedure 338, 491
 - marking stored procedures for replication 344
 - marking stored procedures in a warm standby active database 505
 - using for applied function 338
 - using for request function 342
- sp_setreptable system procedure
 - marking tables in a warm standby active database 505
- sp_stop_rep_agent command 114, 126
- SQL Server
 - login names for LTM access 707
 - truncation point 698
- sql.ini files 79
- sqm_recover_segs configuration parameter 553
- sqm_write_flush configuration parameter 553, 554
- SQT cache 590
- sqt_init_read_delay configuration parameter 553
- sqt_max_cache_size configuration parameter 553, 569
- sqt_max_read_delay configuration parameter 553
- SSL 242
 - certificate authority 242
 - enabling on Replication Server 245
 - on Replication Server 243
 - requirements for 244
 - setting up 244
 - trusted roots file 243
- SSL handshake 243
- SSL security 242
- Stable Queue Manager thread (SQM) 546
 - detecting loss during stable queue rebuild 668
 - handling losses 671
 - loss detection after log recovery 673
- Stable Queue Transaction thread (SQT) 546
- stable queues 555
 - atomic materialization 353
 - described 44
 - detecting losses 668
 - disk files 47
 - DSI loss 668
 - handling partition failure 635
 - off-line rebuild from database logs 666
 - online rebuild 665
 - rebuilding 665
 - requirements 46
 - for routes 148
- standalone mode
 - Replication Server 640, 665
- standby database 480
 - adding 498
 - monitoring status of add 514
 - switching to 506
- star configuration
 - described 144
- starting
 - LTM 700
 - Replication Server 94
 - Sybase Central 60
- stats_daemon_sleep_time configuration parameter 553, 603
- stats_flush_rssd command parameter 602
- stats_reset_afterflush command parameter 602
- stats_sampling command parameter 602
- status
 - verifying data servers 429
 - verifying RepAgents 429
 - verifying Replication Servers 429
- status bar 68–70
- status information
 - topology 86
- stopping Sybase Central 60
- stored procedure subscriptions
 - as objects in Sybase Central 83
- stored procedures
 - as objects in Sybase Central 83
 - dropping 691
 - marking for replication using sp_setreplicate 688
 - marking for replication using sp_setrepproc 343
 - publications 347
 - rs_delexception 624
 - rs_helpclass 473
 - rs_helppdb 194
 - rs_helperror 619
 - rs_helpexception 622
 - rs_helpfstring 473
 - rs_helpprep 299

- rs_helprepdb 390
- rs_helproute 166
- rs_helpsub 390
- rs_helpuser 213
- rs_init_erroractions 616
- rs_mk_rsids_consistent 656
- rs_update_lastcommit 173
- sts_cachesize configuration parameter 554
- sts_full_cache configuration parameter 554
- sub_daemon_sleep_time configuration parameter 554
- sub_sqm_write_request_limit configuration parameter 186, 554
- subscribe to truncate table clause 395
- subscribing
 - to data in warm standby databases 537
 - to function replication definitions 344
 - to replicated tables 350
- subscriptions
 - primary fragments 12
- subscription dematerialization
 - methods 379
 - phases 365
 - processing 363
 - with purge 363
- subscription materialization 186
 - defined 33
 - methods 366
 - phases 364
 - text and image columns 384
- subscription materialization queue
 - defined 45
- subscription migration
 - described 547
 - rs_address columns 389
- subscription resolution engine (SRE) 415, 547
- subscription retry daemon (dSUB) 549
- subscriptions
 - adding to a publication subscriptions 397
 - bitmap 387
 - commands for managing 368
 - comparing after restoring backups 654
 - defined 350
 - displaying 390
 - dropping 203, 378, 421
 - login names and password dependencies 203
 - overlapping 147
 - permissions for creating 208
 - preparations for creating 366
 - primary fragments 9
 - re-creating after backups 660
 - removing rows manually 380
 - requirements 366
 - restrictions in warm standby applications 537
 - user permission requirements 208
 - verifying consistency 390
- Sun Cluster HA 717, 719
 - references 717
- suspect subscriptions 539
- suspend connection command 178, 231, 621
- suspend log transfer command 132, 708
- suspend route command 154
- suspending
 - database connections 177
 - log transfer 131
 - LTM's 131
 - routes 154
- switch active command
 - during atomic materialization 539
 - during subscription dematerialization 540
 - during subscription materialization 538
- Sybase Central 91
 - creating an object in 70
 - for Windows 54
 - Java edition 54
 - managing replication systems with 75
 - menus 67
 - navigating in 61
 - online help 57
 - Replication Server views 81
 - shortcut menus 68
 - stopping 60
 - toolbar 68
 - using publications 310
- Sybase Central window
 - customizing 64
- Sybase Replication Agents
 - as objects in Sybase Central 82
- syntax statements
 - conventions xxv
- sysadmin dropdb command 115, 116, 193
- sysadmin dropldb command 524

Index

- sysadmin droprs command 114
- sysadmin erssd, command 106, 107
- sysadmin purge_route_at_replicate command 164
- sysadmin restore_dsi_saved_segments command 635
- system functions
 - rs_dumpdb 637
 - rs_dumptran 637
- system functions, list of
 - with function-string class scope 440
 - with replication definition scope 442
- system procedures
 - replicating 422
 - sp_helpcounter command 605
 - sp_reptostandby 407
 - sp_setreplicate 284, 688
 - sp_setreproc 344, 505
 - sp_setreptable 276, 505
- system tables
 - described 30
 - rs_config 100
 - rs_databases 194
 - rs_diskaffinity 595
 - rs_exceptscmd 622
 - rs_exceptshdr 622
 - rs_idnames 193, 523
 - rs_lastcommit 188
 - rs_statcounters 598, 605
 - rs_statdetail 598
 - rs_statrun 598
 - rs_systext 622
- system transactions 626

T

- Table Properties dialog box 72
- table replication definitions
 - as objects in Sybase Central 82
- table subscriptions
 - as objects in Sybase Central 83
- tables
 - as objects in Sybase Central 82
 - creating for replication 248
 - procedure for replicating 251
 - subscription requirements 367
- tabs

- help contents 58
- help search 59
- index contents 59
 - on dialog boxes 72
 - on property sheets 72
- testing
 - Replication Server components 427
 - Replication Server connections 427
- text datatype
 - changing replication for 288
 - LTM shutdown and 289
 - overview of replication 283
- threads
 - described 544
 - displaying for replication system 430
 - distributor (dist) 546
 - DSI executor 548, 570
 - DSI scheduler 45, 548, 570
 - in primary Replication Server described 549
 - in primary replication server described 544
 - for parallel DSI 566
 - RS user 549
 - RSI 46, 548
 - RSI user 550
 - Stable Queue Manager (SQM) 545
 - Stable Queue Transaction (SQT) 546
 - USER 549
- threads, miscellaneous 549
- threshold levels
 - setting and using for partitions 432
- timestamp
 - and foreign LTMs 674
 - qid 674
- toolbar 68–70
 - buttons 68
 - hiding 68
 - Sybase Central 68
- toolbar button help 59
- toolbar controls
 - Back 69
 - Connect 69
 - Copy 69
 - Cut 69
 - Delete 69
 - Details 69
 - Disconnect 69

- Drop-down hierarchy 69
- Large Icon 69
- List 69
- Next 69
- Paste 69
- Properties 69
- Small Icon 69
- Up 69
- Tools menu 67
- tooltips 59, 69
- topic help 57
- topology
 - edges 85
 - folders 85
 - status information 86
 - views 81, 84
- Transaction Delivery module (TD) 547
- transaction names, default 579
- transaction size 590
- transactions
 - exceptions handling 620
 - handling suspended 177
 - handling with Replication Server 43
 - large 571
 - loading log dumps 674
 - modifying data in multiple data servers 48
 - processing with parallel DSI threads 566, 591
 - reasons for failure 620
 - serialization methods 572
 - skip transaction clause 188
 - small 571
 - timestamp 674
- Transact-SQL commands
 - dump database 637
 - dump transaction 637
 - set replication off 522
 - set triggers off 522
- triggers
 - configuring in standby databases 522
- truncate table command 626
 - enabling replication of 398
 - RCL 486
 - Transact-SQL 395
- truncated database logs, recovering 642, 645
- truncation of Adaptive Server database logs 49
- trusted roots file 243

U

- unified_login 227
- Up control 69
- updating function strings 466
- upgrading routes 164
- user stored procedures
 - replicating 423
- USER thread 549
- user-defined functions
 - adding parameters 690
 - associating replicated stored procedures with 689
 - described 438
 - dropping 691
 - managing 688
 - mapping to a different stored procedure name 691
 - specifying a non-unique function name 693
- users, displaying replication system 213
- users, ERSSD 108

V

- validate publication command 311, 313, 315
- validate subscription command 369, 395, 399
- variables
 - function strings 42, 462
 - modifiers 462
 - system-defined 462
- verify password clause 205
- verifying translations 330
- version daemon (dVERSION) 549
- version number for route 164
- versions, replication system 18
- View menu 67
- viewing object properties 71
- views
 - logical 81
 - physical 81
 - topology 81, 84

W

- wait for create standby command 505
- wait for switch command 511
- warm standby applications

Index

- comparing methods 485
- database connections 481
- databases 481
- effects of switching to the standby database 509
- forcing replication of DDL commands 493
- logical connections 481
- monitoring 514
- MSA advantages 406
- physical connections 481
 - for a primary database 524
 - for a replicate database 526
- restrictions 483
- setting up databases 494, 521
- switching to the standby database 506
- switchover with MSA 412
- tables with the same name 491
- turning off replication 493
- using MSA 409
- warm standby, alter table command support 532
- where clause
 - altering 317
 - for create subscription command 370
 - limit to expressions 1024 columns 536
 - operators 314
 - syntax 314
 - used in articles 314
- wide columns 270
- wide data 270
- wide messages 271
- with nowait clause 178
- with primary table named clause 259
- with purge clause 401
- with replicate table named clause 259
- without holdlock clause 374
- without purge clause 400
- wizards 71
- write operations 554
- writetext function string output templates 476
- writing directly to media 554