



Administration Guide

OpenSwitch

15.0

DOCUMENT ID: DC20191-01-1500-01

LAST REVISED: December 2005

Copyright © 1999-2005 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, the 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, Adaptive Warehouse, Afaria, Answers Anywhere, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Translator, APT-Library, AvantGo Mobile Delivery, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BizTracker, ClearConnect, Client-Library, Client Services, Convoy/DM, Copernicus, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DataWindow .NET, DB-Library, dbQueue, Developers Workbench, DirectConnect, DirectConnect Anywhere, Distribution Director, e-ADK, E-Anywhere, e-Biz Impact, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, M2M Anywhere, Mach Desktop, Mail Anywhere Studio, Mainframe Connect, Maintenance Express, Manage Anywhere Studio, M-Business Channel, M-Business Network, M-Business Server, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, mFolio, Mirror Activator, MySupport, Net-Gateway, Net-Library, New Era of Networks, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Client, Open Client/Connect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PocketBuilder, Pocket PowerBuilder, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, RemoteWare, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report-Execute, Report Workbench, Resource Manager, RFID Anywhere, RW-DisplayLib, RW-Library, S-Designor, SDF, Search Anywhere, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SOA Anywhere, 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, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase IQ, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, Syber Assist, SybFlex, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, TradeForce, Transact-SQL, Translation Toolkit, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, 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, and XP Server are trademarks of Sybase, Inc. 06/05

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

About This Book	vii
 CHAPTER 1	
Overview	1
What is OpenSwitch?	1
Connection management	2
Connection migration	2
Failure detection and recovery	2
Connection termination	3
Suspending and resuming connections	3
Server pools and routing	4
Mutually-aware OpenSwitch server support.....	8
External coordination using coordination modules	9
Resource governing	11
User messaging	12
Dynamic SQL support	12
International character sets	13
Configuration	13
Standalone GUI configuration	13
Dynamic configuration	14
Deployment issues	14
Connection context.....	14
Deadlocks and RPCs	15
Temporary tables and cursors.....	15
Performance	15
Number of user connections	16
Java-based jConnect client applications	16
 CHAPTER 2	
Concepts and Procedures.....	19
Defining servers	19
Defining a server	19
Server state	20
Server status and existing conditions.....	21
Listing available servers	22

Changing the status of a server	22
Setting server options.....	22
Defining pools	23
Defining a pool	24
Routing and collisions	24
Pools and servers.....	25
Pools and connections	25
Pool states.....	26
Listing available pools	27
Changing pool status.....	28
Managing connections and threads	28
Establishing an outgoing connection.....	29
Forwarding queries and results	32
Monitoring connection state	32
Managing switch requests.....	33
Using connection caching	34
Managing failures	38
Failure detection.....	38
Deadlock messages	39
Working with client-side cursors.....	39
Cursors within dynamic SQL	40
Failover handling	40
Cursor repositioning	41
Enabling SSL support	42

CHAPTER 3	Starting and Stopping OpenSwitch and RCMs.....	43
	Starting and stopping OpenSwitch on UNIX	43
	Starting and stopping OpenSwitch on Windows	44
	Using encrypted user names and passwords	48
	Using command line options	49
	Starting and stopping the RCM from OpenSwitch	54
	Requirements	55
	Configuring an RCM to start automatically from OpenSwitch ..	55

CHAPTER 4	Using the Configuration File.....	59
	Introduction	59
	Editing the OpenSwitch configuration file.....	60
	Using wildcards	60
	Creating or editing a configuration file.....	62
	Manually editing configuration options	63
	[CONFIG]	64
	[SERVER].....	83
	[POOL]	86

	[LIMIT_RESOURCE]	89
	[COMPANION]	91
CHAPTER 5	Using Mutually-aware OpenSwitch Servers	93
	Introduction	93
	Requirements	94
	Installation	94
	Configuration and use	95
	Configuring OpenSwitch servers to be mutually aware	99
	Configuration file parameters	99
	Mutually-aware configuration table	104
	Configuration data precedence	107
	OpenSwitch mutually-aware operations	109
	Active Adaptive Server failover	109
	Failback	110
	Invoking custom and manual scripts	111
	Overview	111
	CMON_FAIL_ACTION	119
	CMP_FAIL_ACTION	120
	NET_FAIL_ACTION	121
	SVR_FAIL_ACTION	122
CHAPTER 6	Registered Procedures	125
	Invoking registered procedures	126
	Remote procedure call invocation	127
	Direct invocation	127
	rp_cancel	128
	rp_cfg	130
	rp_debug	133
	rp_dump	137
	rp_go	137
	rp_help	139
	rp_kill	141
	rp_msg	143
	rp_pool_addattrib	145
	rp_pool_addserver	147
	rp_pool_cache	152
	rp_pool_create	154
	rp_pool_drop	157
	rp_pool_help	158
	rp_pool_remattrib	160
	rp_pool_remserver	162
	rp_pool_server_status	163

rp_pool_status.....	165
rp_rcm_connect_primary	169
rp_rcm_list.....	170
rp_rcm_shutdown.....	171
rp_rcm_startup	172
rp_replay	174
rp_rmon.....	179
rp_server_help	180
rp_server_status	181
rp_set	185
rp_set_srv	188
rp_showquery.....	189
rp_shutdown.....	190
rp_start	191
rp_stop	192
rp_switch	195
rp_traceflag	198
rp_version	200
rp_who	201
 CHAPTER 7	
Notification Procedures	205
Introduction	205
Using notifications	205
Notification registered procedures	206
np_req_srv	206
np_switch_start	208
np_switch_end	208
 Index	211

About This Book

Audience

This book is for OpenSwitch version 15.0 system administrators and assumes that you have:

- General knowledge of your operating system platform
- Familiarity with platform-specific commands used to manipulate the software and hardware
- General knowledge of Sybase™ servers
- General knowledge of failover systems

How to use this book

This document describes OpenSwitch version 15.0 administration and contains the following chapters:

- Chapter 1, “Overview” – describes OpenSwitch functionality, features, and deployment issues.
- Chapter 2, “Concepts and Procedures” – covers concepts, terminology, and procedures.
- Chapter 3, “Starting and Stopping OpenSwitch and RCMs” – describes how to start and stop OpenSwitch from the command line, and provides an easy-reference table of flags used to enable Open Server™ debugging messages.
- Chapter 4, “Using the Configuration File” – describes the sections of the configuration file.
- Chapter 5, “Using Mutually-aware OpenSwitch Servers,” describes how to configure and use mutually-aware OpenSwitch servers.
- Chapter 6, “Registered Procedures” – describes how to invoke registered procedures directly or via remote procedure calls, and provides reference pages for each registered procedure.
- Chapter 7, “Notification Procedures” – describes a special kind of registered procedure that can be used by applications outside Open Client™ to be notified when certain events occur within OpenSwitch.

Related documents

OpenSwitch documentation The following documents are available on the Sybase Getting Started CD in the OpenSwitch 15.0 product container:

- The OpenSwitch installation guide explains how to install the OpenSwitch software.
- The OpenSwitch release bulletin contains last-minute information not documented elsewhere.

OpenSwitch online documentation The following OpenSwitch documents are available in PDF and Eclipse format on the OpenSwitch 15.0 SyBooks CD:

- *What's New?* describes new features in the product.
- The *OpenSwitch Administration Guide* explains how to administer OpenSwitch and how to reconfigure the product after installation.
- The *OpenSwitch Coordination Module Reference Manual* describes how to develop and use OpenSwitch coordination modules.
- The *OpenSwitch Error Message Guide* explains how to troubleshoot problems that you may encounter when using OpenSwitch, and provides explanations of error messages.

Other sources of information

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

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

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

Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Select Products from the navigation bar on the left.
- 3 Select a product name from the product list and click Go.
- 4 Select the Certification Report filter, specify a time frame, and click Go.
- 5 Click a Certification Report title to display the report.

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

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

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

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

- 1 Point your Web browser to the Sybase Support Page at <http://www.sybase.com/support>.
- 2 Select EBFs/Maintenance. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
- 3 Select a product.
- 4 Specify a time frame and click Go.

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

Conventions

The following style conventions are used in this manual:

Commands for both the C shell and the Bourne shell are provided in this document, when they differ. The initialization file for the C shell is called *.cshrc*. The initialization file for the Bourne shell is called *.profile*. If you are using a different shell, such as the Korn shell, refer to your shell-specific documentation for the correct command syntax.

Key	Definition
commands and methods	Command names, command option names, utility names, utility flags, Java methods/classes/packages, and other keywords are in lowercase Arial font.
<i>variable</i>	Italic font indicates: <ul style="list-style-type: none">• Program variables, such as <i>myServer</i>• Parts of input text that must be substituted, for example: <i>Server.log</i>• File names
File Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File Save indicates “select Save from the File menu.”
package 1	Monospace font indicates: <ul style="list-style-type: none">• Information that you enter in a GUI interface, a command line, or as program text• Sample program fragments• Sample output fragments

Accessibility features

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

OpenSwitch version 15.0 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

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

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

If you need help

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

This chapter introduces important concepts about OpenSwitch.

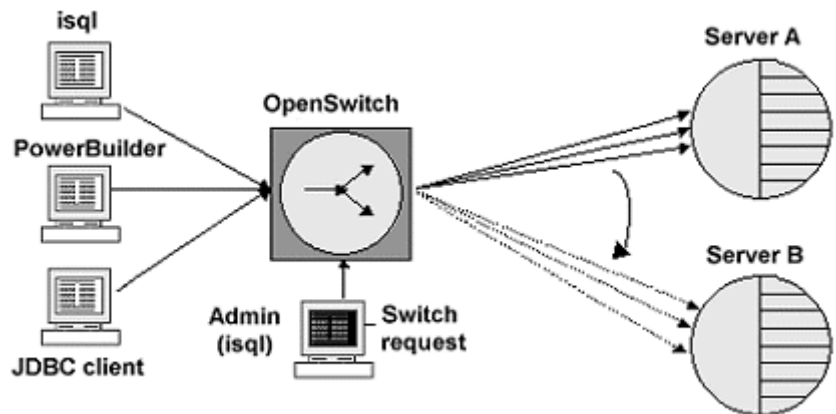
Topic	Page
What is OpenSwitch?	1
Connection management	2
Configuration	13
Deployment issues	14

What is OpenSwitch?

OpenSwitch is a Sybase Open Server gateway placed between client connections such as isql, or any application developed using Sybase Open Client, ODBC, or JDBC libraries, and two or more Adaptive Servers®.

OpenSwitch transparently transfers incoming connections to Sybase server products, such as an Adaptive Server or another Sybase Open Server application (including another instance of OpenSwitch), either manually in response to an administrative request, or automatically in response to an Adaptive Server failure.

Figure 1-1: Basic OpenSwitch functionality



Any OpenSwitch connection switch remains transparent to the client application, and does not require disconnecting and reconnecting. Client applications are presented with a view of one stable connection while behind the scenes, servers are started and stopped for maintenance, failover, batch processing, and so on.

Connection management

This section describes how OpenSwitch manages connections between client applications and Audiotapes Servers and includes references to more detailed information.

Connection migration

Each incoming client application connection is loosely coupled via an outgoing connection to a remote Adaptive Server, allowing OpenSwitch to transparently replace the outgoing connection with a connection to any other server without disturbing the client connection. OpenSwitch attempts to track and restore as much connection state information as possible on each client, such as current database context and transaction state, to ensure that no connection is disturbed while information is being actively transferred between the client and the remote Adaptive Server. See “Managing connections and threads” on page 28 for more information.

Failure detection and recovery

OpenSwitch monitors several important aspects of communication between an incoming client connections and remote Adaptive Servers:

- Transaction state – whether the connection is in the middle of an open transaction.
- Communications state – whether the connection is actively communicating with the Adaptive Server.
- Connection state – whether the connection is currently established with the Adaptive Server.

If the connection to the remote Adaptive Server is shut down, or the connection is killed, OpenSwitch automatically transfers the connection to the next available Adaptive Server.

At the time of failure, if the client connection is either actively communicating with the Adaptive Server (based upon the communications state), or is involved in an open transaction (based upon the transaction state), the client is issued a “deadlock” message indicating that the client should reissue the last transaction sent. For example:

```
Msg 1205, Level 13, State 0
```

```
Server 'OpenSwitch'
```

```
Your command (process id #8) was deadlocked with another process and has been  
chosen as deadlock victim. Re-run your command
```

If the client connection is idle, the switch to the failover server is completely transparent.

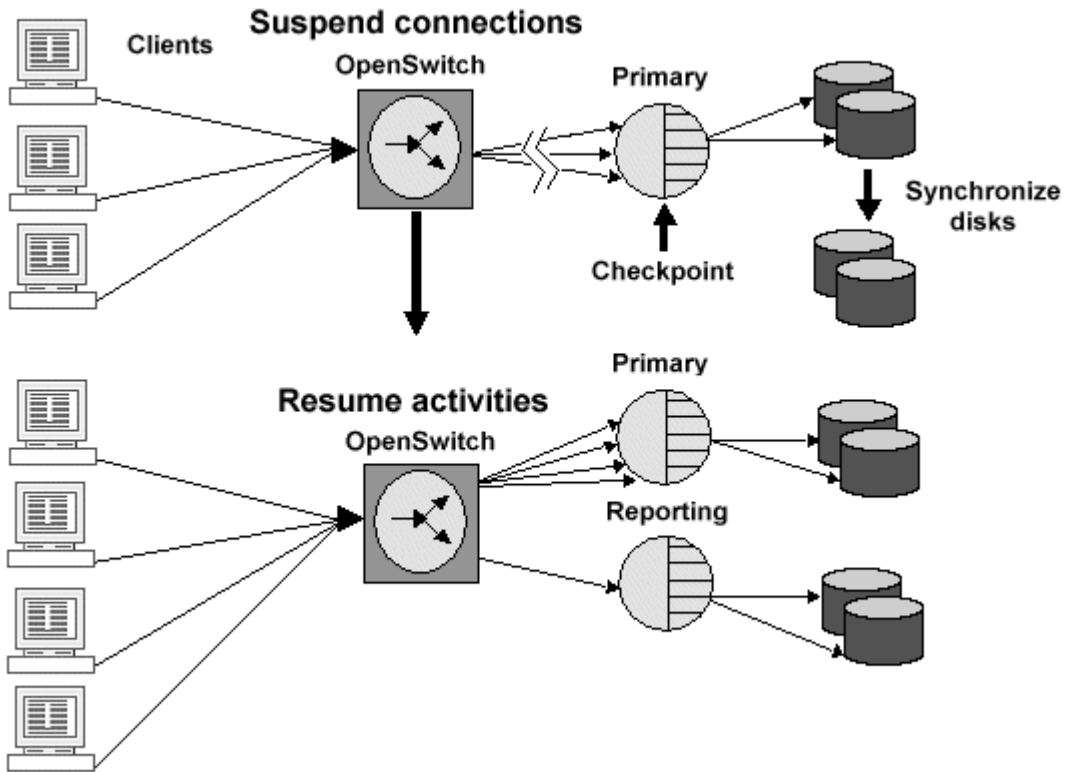
See “Failure detection” on page 38 for more information.

Connection termination

When OpenSwitch begins switching connections, all idle connections are immediately switched to the secondary Adaptive Server, and all busy connections are tagged to be switched when they go idle. An internal service thread within OpenSwitch tracks busy connections that are tagged for switching. If the connection remains busy for a period of time that is defined by the system administrator, the connection is terminated or canceled. See “Managing switch requests” on page 33.

Suspending and resuming connections

Under some circumstances, you may need to shut down a remote server for a period of time to perform some maintenance task. For example, to synchronize two servers, you must suspend all of that server’s activity.

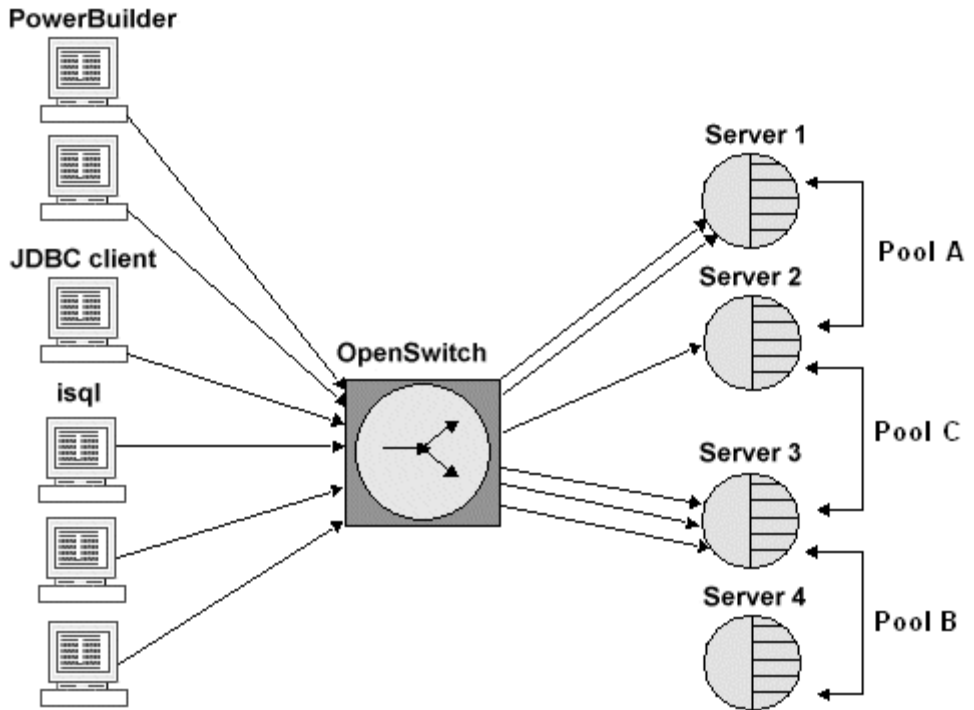
Figure 1-2: Suspend and resume example

OpenSwitch provides a mechanism to suspend a group of connections (either by pool, server, or individual connection) so that no activity is performed on the server until the connections are resumed.

See `rp_stop` on page 192 and `rp_start` on page 191 for instructions.

Server pools and routing

OpenSwitch allows you to create one or more logical groups of servers (each group containing any number of Adaptive Servers), and to route individual connections to a given pool of servers using the incoming user name, the host name, or the application name of the connection as illustrated in Figure 1-3 on page 5. This ensures that administrative requests or automatic failover occurs at the same time and in the same way for all application users.

Figure 1-3: Routing example

In this example, all incoming PowerBuilder® connections are routed to Pool A, for which the primary server is Server 1 and the failover server is Server 2. The JDBC client connections are routed to Pool C, for which the primary server is Server 2 and the failover server is Server 3. All isql connections are routed to Pool B, for which the primary server is Server 3 and the failover server is Server 4.

See “Defining pools” on page 23 for more information about using server pools and routing.

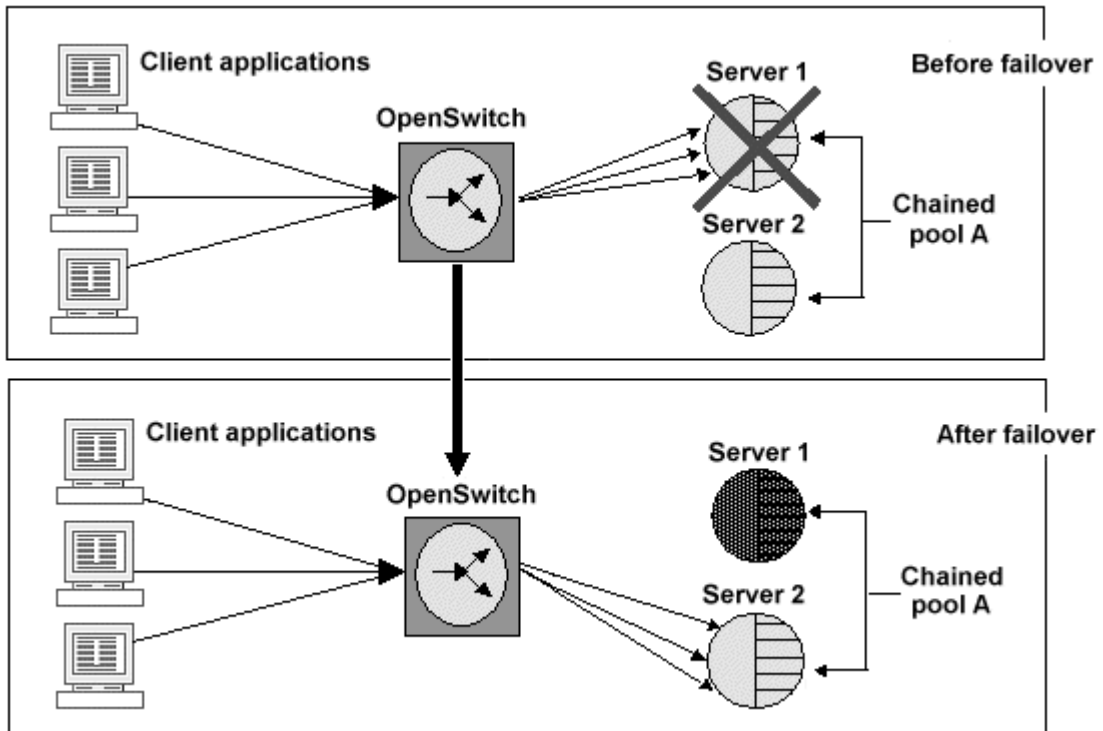
Load balancing and chaining

When you define a pool, you can assign it one of two modes—chained or balanced. The mode determines how connections are assigned or routed to servers in the pool and how connections are managed during failover.

Chained mode

In chained mode, all connections are routed to the first server defined within the pool that has a status of “UP” or “LOCKED,” and administrative switch requests or automatic failover sends all connections to the next server in the pool, as illustrated in Figure 1-4. The order in which servers are accessed is the same order in which they were defined in the pool.

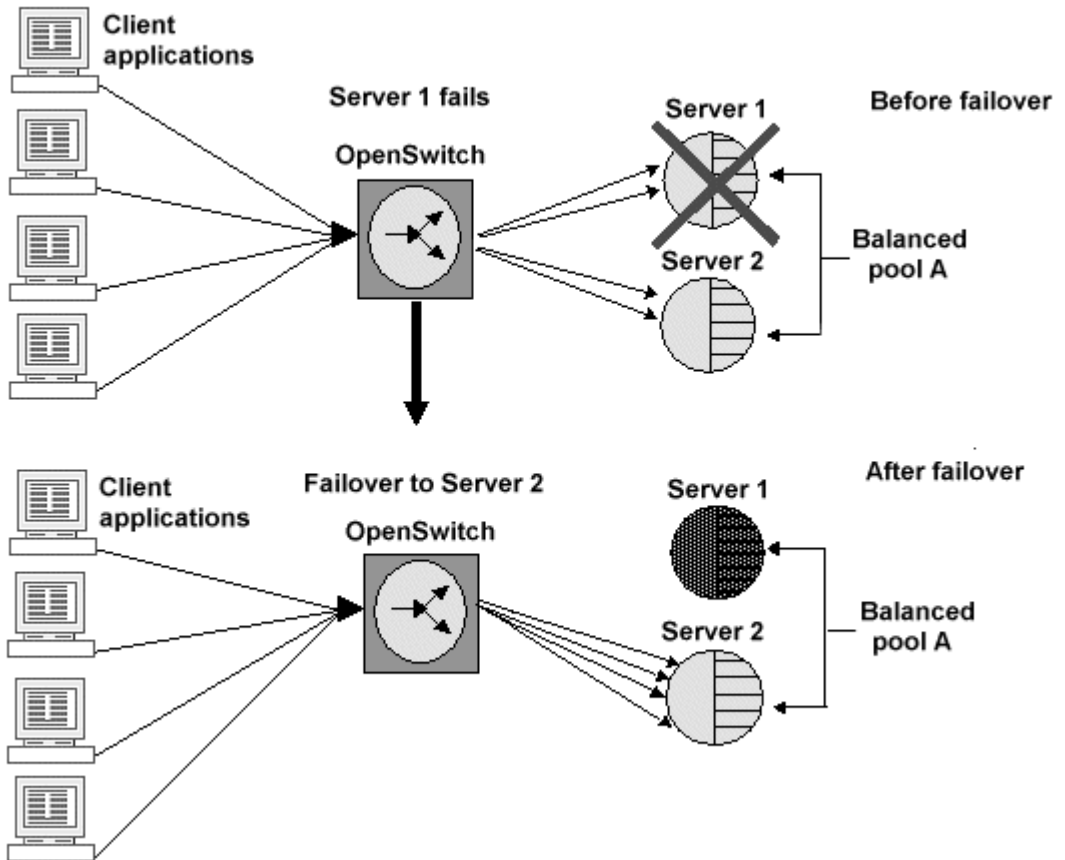
Figure 1-4: Chained mode failover



Balanced mode

In balanced mode, incoming connections are simultaneously routed to all servers in a pool that have a status of “UP” or “LOCKED” until a server fails, effectively balancing the user load across all servers. When a server fails, all connections on the failed server are redistributed, in round-robin fashion, among the remaining servers, as illustrated in Figure 1-5 on page 7. The order in which servers are assigned is determined by the order in which they are defined in the pool.

Figure 1-5: Balanced mode failover



In both chained and balanced modes, any failed attempt to log in to, or fail over to, a server that has a state of “UP” routes the client connection to the next server in the pool with a status of “UP” or “LOCKED.” If all servers within the pool are exhausted, incoming connections are disconnected from OpenSwitch, and a message indicating that no servers are available is returned.

Connection pools and caching

With its default behavior, OpenSwitch uses approximately twice the amount of time normally taken by an application to connect to an Adaptive Server. After a connection has been established to OpenSwitch, another outgoing connection must be established to the remote Adaptive Server.

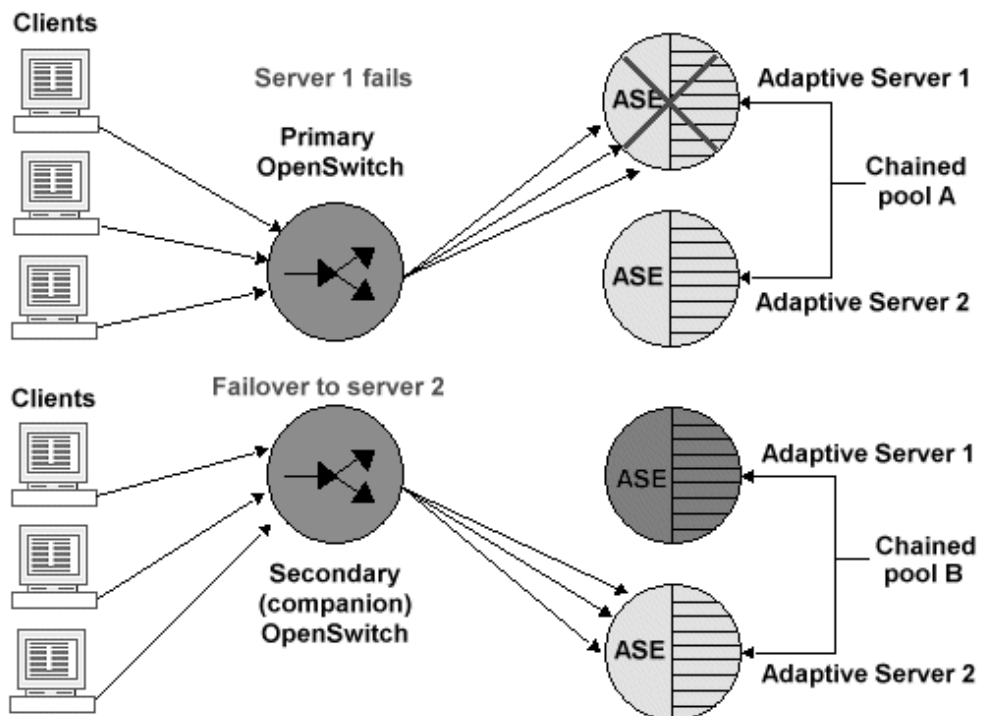
To reduce this overhead, OpenSwitch provides connection pool caching, which allows you to retain connections from specific applications, users, or clients following their disconnection from the client. If the same client reconnects before a specified caching timeout period, the cached connection is reassigned.

Connection caching can benefit applications that rapidly create and drop connections during normal work, such as Web applications or queries through Adaptive Server site handlers. See “Using connection caching” on page 34.

Mutually-aware OpenSwitch server support

OpenSwitch supports a redundant environment with two “mutually-aware” OpenSwitch servers that serve the same pools of two database servers. Each OpenSwitch server is aware of the other OpenSwitch server and whether the connections and Adaptive Server for which that server is responsible are running or not, which prevents a OpenSwitch from being a single point of failure.

Figure 1-6: Mutually-aware OpenSwitch servers



See Chapter 5, “Using Mutually-aware OpenSwitch Servers,” detailed information about how to configure this functionality.

External coordination using coordination modules

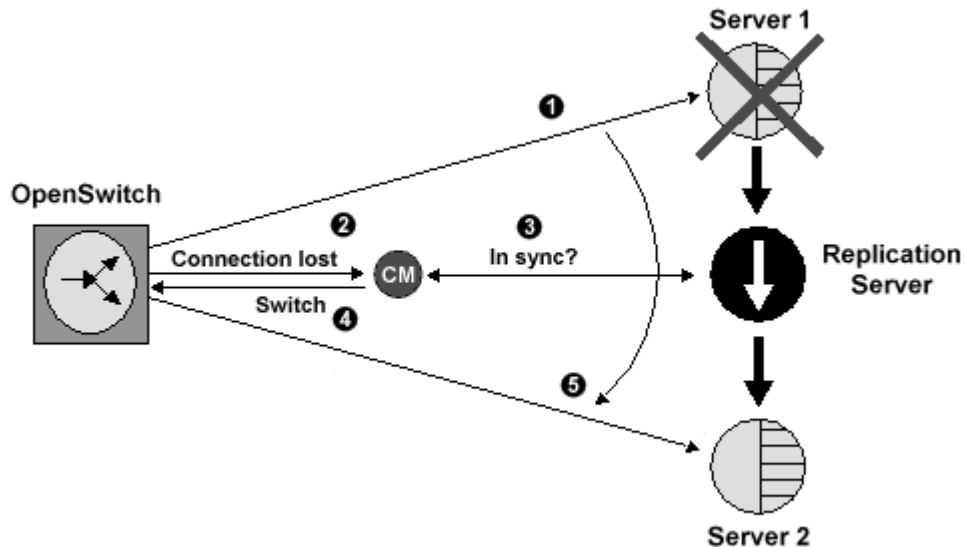
The default behavior of OpenSwitch is to automatically attempt to migrate individual failed connections as they fail. That is, if a single connection fails, it is immediately migrated to the next available server according to the mode of the pool in which the connection resides. However, you may want to coordinate the switching process around certain operation or business requirements.

For example, rather than immediately migrating a connection to the next available Adaptive Server, you may first want to attempt to reconnect to the failed server to ensure that it has failed. Or, you may want to switch all connections to the server if a single connection fails unexpectedly.

More importantly, you may need to coordinate the switching process with an external high-availability (HA) solution such as Sybase Replication Server[®]. In this case, a failover should not occur until the HA service has completed the necessary steps to bring the backup server online, such as waiting until replication queues are synchronized between servers.

For these situations, OpenSwitch provides a simple application programming interface (API) for developing an external coordination module (CM). When connected to an OpenSwitch, a coordination module receives event notifications based on connection state changes (for example, a user attempts to log in, or a connection is lost to a server), and is expected to respond to OpenSwitch, informing it of any actions to take, as illustrated in Figure 1-7 on page 10.

Figure 1-7: Coordination module example



In this example:

- 1 Server 1 goes down unexpectedly; for example, due to a power outage or an explicit shutdown.
- 2 As soon as the connection is lost, the coordination module receives a message indicating which connection was lost, and to which Adaptive Server that connection was communicating. The connection that was lost suspends within the OpenSwitch until the coordination module responds with what should happen to the connection.
- 3 The coordination module now communicates with the high-availability solution, in this case, a replication agent, to ensure that Server 2 is in a state that all users can rely on, such as ensuring that all transactions have been successfully migrated through the replication agent. The coordination module could, at this point, attempt to automatically recover Server 1 before attempting to switch users to Server 2.
- 4 The coordination module responds to the OpenSwitch server that all connections that are using Server 1 should now switch to the next available Adaptive Server, in this case, Server 2.
- 5 All connections are switched, as requested by the coordination module, to the next available server. Connections are issued a “deadlock” message, if necessary.

Because the coordination module can intercept and respond to every connection state change, including client login, you can also use it to override any of the built-in OpenSwitch pooling and routing mechanisms with application- or business-specific logic.

If the OpenSwitch is configured to use a coordination module and one is not available when a connection changes state, the connection suspends until a coordination module comes online, at which time all pending notifications are delivered.

See the *OpenSwitch Coordination Module Reference Manual* for instructions about developing and using OpenSwitch coordination modules.

Resource governing

Often, when attempting to coordinate a combined decision-support system (DSS) where users read the data in the database but never write to it, and an online transaction processing (OLTP) environment where application end users write to the back-end database, DSS users tend to use large amounts of resources on the remote Adaptive Server.

OpenSwitch allows you to automatically cancel or terminate connections that overuse Adaptive Server by setting the *RMON* parameter to 1 in the [CONFIG] section of the OpenSwitch configuration file, then using the [LIMIT_RESOURCE] section to set the maximum amount of a resource that can be consumed by a given connection.

Note You must set the *RMON* parameter in the [CONFIG] section to have the options in the [LIMIT_RESOURCE] section take effect.

This allows you to define resource limits in terms of user name, application name, and host name of incoming user connections, so you can define different resource limits based on the type of incoming connection. You can also configure the resource monitor to either forcibly terminate offending connections, or cancel the current query and return control to the client application with a notification of the reason for the cancellation.

See “[CONFIG]” on page 64 and “[LIMIT_RESOURCE]” on page 89 for more information.

User messaging

Similar to the standard UNIX `wall` or `write` command, OpenSwitch can selectively broadcast administrative messages to client connections that are being actively managed. These messages display as informational server messages during the next executed command batch.

See “Notification Procedures” on page 205.

Dynamic SQL support

Dynamic SQL is the process of generating, preparing, and executing SQL statements at runtime using commands initiated by the Client-Library™ `ct_dynamic` routine.

These dynamic SQL statements are supported in OpenSwitch:

- `CS_PREPARE` – OpenSwitch stores prepared statements in an internal list. If the connected Adaptive Server fails, OpenSwitch fails over to the next Adaptive Server in the pool and re-prepares any existing prepared statements. If `CS_DEALLOC` is used for a particular statement, that statement is removed from the list and is not failed over.
- `CS_EXECUTE`, `CS_EXECUTE_IMMEDIATE`, `CS_DESCRIBE_INPUT`, `CS_DESCRIBE_OUTPUT` – if a failover occurs while these statements are active, a 1205 error is returned to the client, and the client must reissue these statements.
- `CS_CURSOR_DECLARE` – if a dynamic cursor is active during a failover, the cursor is redeclared to the new server and the cursor is positioned to the point of the last successful fetch. A 1205 error is issued to the client, and the client must reissue the last fetch.

See Chapter 8, “Using Dynamic SQL Commands” in the *Open Client Client-Library/C Programmer’s Guide* for more information.

There are two methods of executing dynamic SQL commands—`execute-immediate`, and `prepare-and-execute`.

Execute-immediate

In the `execute-immediate` method, the client application sends the server a `ct_dynamic` command that executes a literal statement.

Execute-immediate is usually used for one-time execution, and does not involve fetchable data. Also, it does not use a unique identifier. Therefore, if a failover were to occur in the middle of its execution, the entire transaction is rolled back, and a 1205 deadlock message is sent to the client application.

Prepare-and-execute

In the prepare-and-execute method, the client application sends the server a sequence of server commands that prepares a statement, and executes it one or more times. The application can send additional commands to query the server for the formats of the statements' input parameters and the result set it returns.

Prepare-and-execute is used for commands that are usually executed multiple times, perhaps with different parameters each time. It can involve fetchable data as well as cursors. It uses a unique identifier that must be preserved during a failover.

International character sets

OpenSwitch supports international character sets. OpenSwitch performs character set conversion of SQL result sets and error messages if the client character set differs from the OpenSwitch default character set.

See the *International Developer's Guide* in the Software Developer's Kit version 15.0 documentation collection for more information.

Configuration

To configure OpenSwitch, use the standalone configuration tool or manually edit the OpenSwitch configuration file.

Standalone GUI configuration

OpenSwitch includes a configuration tool with a graphical user interface (GUI). Access the configuration tool either directly from the OpenSwitch installation program, or by starting it as a standalone application after installation.

See the *OpenSwitch Installation Guide*, Chapter 3, “Configuring OpenSwitch,” for specific instructions.

Dynamic configuration

You can also define all configurable features of OpenSwitch using a text editor to modify the external configuration file, which can be reread without restarting OpenSwitch. This allows for major behavior changes without interrupting user connections.

See the OpenSwitch installation guide for your platform, and “Using the Configuration File” on page 59 for details.

Deployment issues

OpenSwitch has several inherent limitations that may affect different application environments. Use the descriptions of the following issues to determine if OpenSwitch is an appropriate tool for a particular environment.

Connection context

OpenSwitch tracks and restores database context changes per incoming user connections provided that the context changes are made using remote procedure calls (RPCs) or a language call (an explicit `USE` statement). It does not, however, track and restore current character set or language context for a given connection, if the context is changed following the initial connection. This may affect multilingual environments.

Additionally, OpenSwitch cannot track and restore other connection-based context information stored within Adaptive Server, such as session settings. For example, a connection that has issued a `SET SHOWPLAN ON` option resets to `OFF` following an OpenSwitch switch request. However, when these options are set programmatically (via `dboption()` or `ct_options()`), they are properly restored following a switch request.

Deadlocks and RPCs

OpenSwitch uses the Adaptive Server deadlock message (message number 1205) to notify clients when a switch has occurred that has either caused a transaction to be rolled back or has returned an incomplete result set. Under normal circumstances, Adaptive Server also returns a status of -3 from a stored procedure, which indicates that the execution has been halted due to deadlock. OpenSwitch attempts to emulate this behavior. However, for performance reasons, OpenSwitch does not return a status of -3 from stored procedures executed through a SQL language request, such as `exec procedure`.

Temporary tables and cursors

OpenSwitch attempts to restore as much of the original connection context as possible during a switch, however, it cannot directly restore temporary tables or server-side cursors established by the client prior to the switch. For details on handling CT-Lib client-side cursors, see “Working with client-side cursors” on page 39. Ensure that all temporary tables and cursors exist only within single SQL batches or stored procedures submitted by a client.

Performance

Adding an extra tier of processing in any environment adds overhead between the client connections and Adaptive Server. When establishing a connection, the connect time is essentially doubled, as the connection must first be established to OpenSwitch, which in turn establishes a connection to the remote Adaptive Server. This may cause performance difficulties for applications that require the ability to rapidly create and destroy connections. However, under many circumstances, you can address this issue using connection caching. See “Connection pools and caching” on page 7.

Additionally, OpenSwitch must closely monitor all traffic passing between the Adaptive Server and the client connection to detect things such as connection context information (current database context and transaction state), which has an impact on performance, especially when large result sets are involved.

When you run OpenSwitch in full pass-through mode, it creates a pipeline for the requests from the clients to reach the server, and the results from the server to reach the client, without monitoring and processing each individual request and result. This reduces the overhead introduced by OpenSwitch in both directions, and can improve performance significantly. However, it comes at the loss of the ability to perform context failover (specifically, preserving connection state information, especially database context) when a server failure occurs, so it is only recommended for special cases where context failovers are not needed and failovers need not be seamless.

Note For information about configuring “full pass-thru mode” using the *FULL_PASSTHRU* OpenSwitch configuration parameter, see “[CONFIG]” on page 64.

Number of user connections

Because OpenSwitch runs as a single process, it is constrained by the host environment operating system and any limitation on the number of open files per user process. In most environments, the open files allowed per user process is between 1024 to 8192.

Furthermore, OpenSwitch maintains two open connections, one from the client and one to the remote server.

You can address these issues by running multiple instances of the OpenSwitch process.

Java-based jConnect client applications

Because OpenSwitch is based on Open Client-Library, it does not by default relay the Tabular Data Stream™ (TDS) tokens DONEPROC and DONEINPROC from Adaptive Server to the client.

Although this does not affect other Client-Library-based client applications, a Java-based jConnect™ for JDBC™ client application’s behavior may be different connecting through Adaptive Server directly versus connecting through OpenSwitch. For example, when returning from an update table command, the jConnect client application might compute a row update count of zero when it is going through OpenSwitch, even though the update itself was successful and involved several rows.

To achieve the behavior that you get using TDS DONEPROC and DONEINPROC tokens, set the USE_DONEINPROCS option to 1 in the [CONFIG] section of the OpenSwitch configuration file.

Concepts and Procedures

This chapter describes OpenSwitch concepts and provides procedures for performing tasks.

Topic	Page
Defining servers	19
Defining pools	23
Managing connections and threads	28
Managing failures	38
Working with client-side cursors	39
Enabling SSL support	42

Defining servers

In OpenSwitch terminology, a server is a remote application capable of receiving and processing TDS requests (the protocol used by clients to communicate with the OpenSwitch server), and returning results back to client applications; for example, Sybase Adaptive Server Enterprise, Sybase IQ, or any Sybase Open Server application.

A server is made available or visible to OpenSwitch via the configuration file at start-up (specified in the [SERVER] section), or through registered procedures such as `rp_pool_addserver`.

Defining a server

Define a server and its associated state using the OpenSwitch configuration tool or within the [SERVER] section of the OpenSwitch configuration file:

```
[SERVER=SYB_SERV1]
    STATUS=UP
```

Servers are implicitly created when you add them to a pool.

Note OpenSwitch 15.0 includes a configuration tool with a graphical user interface (GUI). Access the configuration tool either directly from the OpenSwitch installation program, or by starting it as a standalone application after installation. See the OpenSwitch installation guide, Chapter 3, “Configuring OpenSwitch” for more information.

See “Using the Configuration File” on page 59 for instructions on manually editing the OpenSwitch configuration file.

Server state

Internally, OpenSwitch maintains a state disposition for each server. The state indicates whether the server is available for use by the connections that it manages. Table 2-1 lists the possible server states:

Table 2-1: Server states

State	Description
PRE_UP	Mutually-aware-specific server status. The server is either in the process of being marked as UP, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to UP on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running. Warning! Do not manually set a server’s status to PRE_UP.
UP	The server is immediately available for use.
PRE_DOWN	Mutually-aware specific server status. The server is either in the process of being marked as DOWN, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to DOWN on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running. Warning! Do not manually set a server’s status to PRE_DOWN.
DOWN	The server is unavailable, and is not considered for use by any new client connections established to OpenSwitch.

State	Description
PRE_LOCKED	<p>Mutually-aware specific server status. The server is either in the process of being marked as LOCKED, or has encountered a problem during that process. doing so. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to LOCKED on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a server's status to PRE_LOCKED.</p> <hr/>
LOCKED	<p>The server is available, but any new incoming connections actively being connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to the client applications to have stopped responding until the pool is unlocked.</p>

Server state changes

OpenSwitch does not determine the state of a server, even if a connection to the server failed. The server state can be assigned only by an administrator or a coordination module via an RPC such as `rp_server_status` or within the configuration file.

A server that has a state of UP remains UP, even if the actual server process is no longer running. New connections attempt to establish connections to the server, even though it is actually unavailable. The connections are automatically routed to the next available server after the initial failure.

You can employ a coordination module to have server state established automatically. See the *OpenSwitch Coordination Module Reference Manual* for instructions on creating and using coordination modules.

Server status and existing conditions

The status of a server is used by OpenSwitch only when an incoming connection is established or when a server is needed for failover. Therefore, changing the state of a server where existing connections have already been established has no effect on the connections. For example, after issuing the following statement, any existing connections to SERVER3 remain connected. However, new connections to OpenSwitch are no longer routed to SERVER3.

```
rp_server_status "SERVER3", "DOWN"
```

Listing available servers

Use `rp_server_status` without any arguments to display available servers and their status. For example:

```
rp_server_status
```

returns:

server	status
-----	-----
SERVER1	UP
SERVER2	UP
SERVER3	LOCKED

See `rp_server_status` on page 181 for more information.

Changing the status of a server

Use `rp_server_status` to change the status of a server at runtime. For example:

```
rp_server_status "SERVER1", "DOWN"
```

returns:

server	status
-----	-----
SERVER1	DOWN

Setting server options

Server options refer to Adaptive Server option settings, which are session-level settings that affect the way Adaptive Server handles results and various aspects of a client connection. For example, when setting the `ROWCOUNT` option, Adaptive Server limits all result sets to the number of rows specified with the option.

To set such options within Adaptive Server, use a language command, an explicit Adaptive Server Enterprise system procedure (sp_dboption), or an Open Client API request (ct_options).

Note See the *Adaptive Server Enterprise Reference Manual: Procedures*, Chapter 1, “System Procedures,” for instructions on using the system procedure sp_dboption. See the *Open Client Client-Library/C Reference Manual* for instructions on using ct_options.

OpenSwitch does not parse the SQL code passed between Adaptive Server and the client. Therefore, it is not aware of options set with either language commands or the SET command. Such option settings are lost when the connection is switched from one server to another.

OpenSwitch is aware of options set using Open Client API calls, and internally tracks the options that are currently set on the connection. These settings are restored correctly on any new servers to which the connection is switched.

Note Sybase strongly recommends that client applications use an Adaptive Server Enterprise system procedure or an Open Client API call rather than language commands to set options.

Defining pools

A pool is a group of servers within OpenSwitch. A pool can contain one or more servers that are treated as a self-contained failover group, so all connections within the group fail over only to servers defined within the group.

A pool can also optionally define the set of connections or connection attributes that it manages. This association between a connection and a pool is determined based on the user name, application name, or connection type.

Defining a pool

Note The order in which pools are defined is important; all connections are routed to the first matching pool according to the *attribute/value* pairs established using `rp_pool_addattrib`. See `rp_pool_addattrib` on page 145 for details.

You define a pool using the OpenSwitch GUI configuration tool during or after installation, or by manually editing the OpenSwitch OpenSwitch configuration file. For example:

```
[POOL=POOL_A:MODE=CHAINED,CACHE=0]
servers:
    SYB_SERV1
    SYB_SERV2
    SYB_SERV3
connections:
    type: client, site
    appname: isql
    username: ^test.*
```

In this example, `POOL_A` contains three servers: `SYB_SERV1`, `SYB_SERV2`, and `SYB_SERV3`. It can be used by regular `client` and `site` handler connections created by `isql`, or any user that starts with “test.”

If you do not supply any “connections” attributes when defining a pool, all connections are candidates for the pool.

Routing and collisions

You can define multiple pools that all match connection properties of a single incoming client connection. For example:

```
[POOL=POOL_A:MODE=CHAINED]
servers:
    ...
connections:
    appname: isql

[POOL=POOL_B:MODE=CHAINED]
servers:
    ...
connections:
```

```
username: test%
```

If the user “test” attempts to connect to OpenSwitch using `isql`, he or she is a candidate for both `POOL_A` and `POOL_B`. In this case, OpenSwitch uses the first matching pool as defined in the configuration file or as listed in `rp_pool_help`.

Pools and servers

Usually, servers and pools are only loosely associated within OpenSwitch; that is, when a server’s state changes, the change is reflected across all pools in which the server is defined. However, you can create a tighter association between a pool and a server by setting the pool-specific server status.

Use any of these procedures to set the pool-specific status of a server:

- Execute `rp_pool_addserver` and specify the status of the server being added to the pool. See `rp_pool_addserver` on page 147.
- Call `rp_pool_server_status` to set a server’s status within a pool. See `rp_pool_server_status` on page 163.
- Specify a server’s status in the `[POOL]` section of the OpenSwitch configuration file. See “[POOL]” on page 86.

When you set a pool-specific server status, that status affects only the specified pool. If you do not explicitly set a server’s status for a specific pool, each server’s status defaults to the general server status, which is defined in the `[SERVER]` section of the OpenSwitch configuration file).

Pools and connections

When an incoming connection is received, the pool name determines the server that the connection should be using. Once the server is determined, the pool name is discarded. No relationship is maintained between the pool name and the server.

Therefore, removing a server from a pool using `rp_pool_remserver "Pool_A", "SYB_SERV3"` does not affect any of the existing OpenSwitch connections that have been routed to `SYB_SERV3` via pool `POOL_A`. This removal causes only `SYB_SERV3` to be removed from consideration by subsequent connections established to OpenSwitch.

Pool states

The state of a pool applies only to connections that are actively requesting a server to be used from the pool during login or during failover. Table 2-2 lists possible pool states:

Table 2-2: Pool states

Status	Description
PRE_UP	<p>Mutually-aware-specific pool status. The pool is either in the process of being marked as UP, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to UP on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_UP.</p> <hr/>
UP	<p>The pool is immediately available for use.</p>
PRE_DOWN	<p>Mutually-aware specific pool status. The pool is either in the process of being marked as DOWN, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to DOWN on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_DOWN.</p> <hr/>
DOWN	<p>The pool is unavailable, and is not considered for use by any new client connections established to OpenSwitch.</p>
PRE_LOCKED	<p>Mutually-aware specific pool status. The pool is either in the process of being marked as LOCKED, or has encountered a problem during that process. doing so. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to LOCKED on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_LOCKED.</p> <hr/>
LOCKED	<p>The pool is available, but any new incoming connections actively being connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to the client applications to have stopped responding until the pool is unlocked.</p>

Status	Description
SUSPENDED	The pool is being suspended by OpenSwitch due to a failure that requires an administrator's manual intervention. See "Invoking custom and manual scripts" on page 111 for more information. The pool blocks on all new connections until <code>rp_go</code> is issued.
	Warning! Do not manually set a pool's status to SUSPENDED.

Listing available pools

To list the set of available pools, enter:

```
rp_pool_help
```

Sample results are:

pool_name	mode	status	block	next_server
POOL_A	CHAINED	UP	0	SYB_SERV1
POOL_B	BALANCED	UP	0	SYB_SERV2
POOL_C	BALANCED	LOCKED	0	SYB_SERV3

To list details about a specific pool, add the name of the pool about which you are inquiring. For example:

```
rp_pool_help "POOL_A"
```

Sample results are:

pool_name	mode	status	block	next_server
POOL_A	CHAINED	UP	0	SYB_SERV1


```
server_name
-----
SYB_SERV1
SYB_SERV3
```


attribute	value
appname	sql
hostname	test.sybase.com

See `rp_pool_help` on page 158 for more information.

Changing pool status

Use `rp_pool_status` to change the status of a pool. For example:

```
rp_pool_status "POOL_A", "LOCKED"
```

returns:

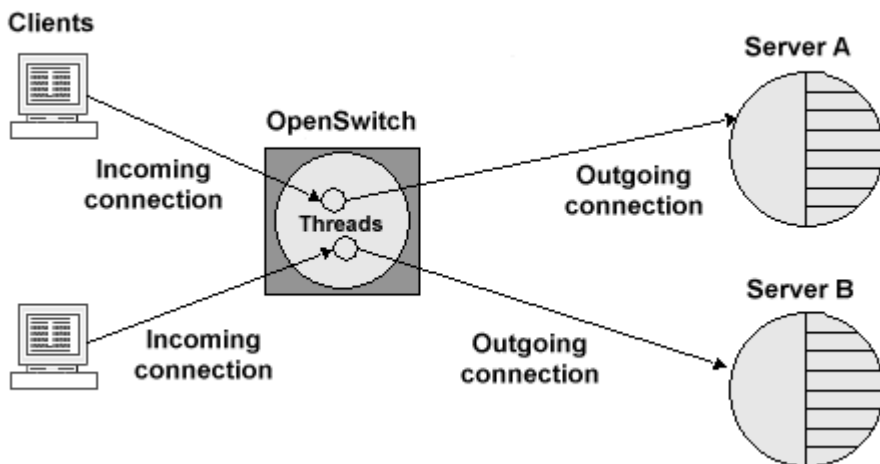
```
pool_name  status
-----
POOL_A     LOCKED
```

See `rp_pool_status` on page 165 for more details.

Managing connections and threads

The primary task of OpenSwitch is to manage incoming user client application connections. OpenSwitch does this by capturing connection information about the client, such as user name and password, and using this information to establish an Open Client connection to the remote database server. This connection is bonded, or attached, to the incoming client connection so that each incoming connection has its own outgoing server connection for the life of the client application session. Each bond between connections is managed by an Open Server thread so that multiple bonds are managed simultaneously.

Figure 2-1: Connections and threads



Each OpenSwitch thread is responsible for these tasks:

- Establishing an outgoing connection
- Forwarding queries and results
- Monitoring connection state
- Managing switch requests

Establishing an outgoing connection

When a client connection is established to OpenSwitch, a thread is spawned to manage this connection. The thread performs these tasks:

- 1 Based on the user name, application name, and host name of the incoming client connection, the thread determines which pool to use. The pool must have a state of either UP or LOCKED. If the pool state is UP, the thread proceeds to step 2. If the pool state is LOCKED, the thread sleeps until the state changes to either UP or DOWN, unless NOWAIT_ON_LOCKED is set to “1” in the OpenSwitch configuration file, in which case the thread returns to the client with a descriptive message, and no connection is established with the data server. See “Defining pools” on page 23.
- 2 The thread requests the name of the next available server from the pool. The choice of server is based on the mode of the pool (CHAINED or BALANCED), as well as the state of the server (UP, DOWN, or LOCKED). A server is considered available only if its state is UP or LOCKED. If it is UP, the thread proceeds to step 3. If it is LOCKED, the thread sleeps until the server state changes to either UP or DOWN, unless NOWAIT_ON_LOCKED is set to 1 in the configuration file, in which case this thread returns to the client with a descriptive message and no connection is established with the data server.
- 3 Using the user name and password of the incoming connection, an outgoing connection is established to the data server, and then bound to the incoming connection.

The following sections outline how the managing thread responds to various failure conditions during this process.

Server unavailable

If the remote server chosen from the pool is unavailable, or if there is no appropriate entry in the `$SYBASE/interfaces` file on UNIX or the `%SYBASE%\sql.ini` file on Windows:

- 1 The name of the next available server is requested from the pool, and a connection is attempted on that server.
- 2 If the connection attempt fails, the name of the next available server in the pool is requested. If there are no remaining servers, the client is disconnected and the login process fails.

Login denied

If the remote server denies a login for any reason, the client connection is removed from OpenSwitch and no failover or attempt to reconnect is tried.

Connection refused

If the Adaptive Server max network packet size configuration parameter is set to 512 (the default), clients connections to OpenSwitch fail and the client receives this error message:

```
The packet size (2048) specified at login time is
illegal. Legal values are between 512 and 512.
```

To correct this problem you can reconfigure the Adaptive Server max network packet size to be 2048, or you can reconfigure the OpenSwitch `MAX_PACKETSIZE` to be 512.

❖ Reconfiguring Adaptive Server *max network packet size* to be 2048

- 1 Set the SYBASE environment variable. In a Command Prompt window, go to `%SYBASE%` on Windows and `$SYBASE` on UNIX and enter:

```
SYBASE.bat (Windows)
```

or

```
SYBASE.csh (UNIX)
```

- 2 Use `isql` to log in to the Adaptive Server as an administrator:

```
isql -Usa -P -S <Adaptive_Server_server_name>
```

- 3 Execute:

```
sp_configure 'max network packet size', 2048
```

```
go
```

- 4 Restart the Adaptive Server.
- 5 Restart OpenSwitch to establish the client connection.

❖ Reconfiguring OpenSwitch MAX_PACKETSIZE to be 512

Remember that `MAX_PACKETSIZE` refers to the maximum size of the TDS packet. This option is used to tune the data throughput across the network and can significantly improve performance when larger packet sizes are used.

- 1 Shut down the OpenSwitch server:
 - a Set the SYBASE environment variable. In a Command Prompt window, go to `%SYBASE%` on Windows and `$SYBASE` on UNIX and enter:

```
SYBASE.bat (Windows)
```

or

```
SYBASE.csh (UNIX)
```

- b Use `isql` to log in to OpenSwitch as an administrator:

```
isql -U<ADMIN_USERNAME>
      -P<ADMIN_PASSWORD>
      -S <OpenSwitch_server_name>
```

- c Execute:

```
rp_shutdown
go
```

- 2 Use a text editor to modify the OpenSwitch configuration file (`<OpenSwitch_server>.cfg`), which is located in `%OPENSWITCH%\config` on Windows and in `$OPENSWITCH/config` on UNIX.

Change the `MAX_PACKETSIZE` value from 2048 (the default) to 512. For example:

```
MAX_PACKETSIZE=512
```

- 3 Restart the OpenSwitch server.

Forwarding queries and results

When a server request is made by the client, usually in the form of a SQL language request, or a call to a registered procedure, the thread receives the query and forwards it to the remote Adaptive Server.

When results are ready from the remote server, the thread returns the results to the client.

If, at any time during this process, a problem is encountered by the thread (such as the loss of the connection to the remote server, or an internal OpenSwitch error), the result set is stopped, and an error condition is returned to the client.

Monitoring connection state

While forwarding client query requests and returning server results, OpenSwitch monitors several aspects of the client connection, including database context, communications state, and transaction state.

Database context

Each time the client connection changes its database context (usually by issuing a USE statement), OpenSwitch captures the context change information and keeps track of the current database in use by the connection. This information is used during a switch or failover to restore the user's database context back to its original state.

Communications state

Each time a client request is received by the thread, the client connection is marked as “busy.” This protects the connection from being switched by an administrative request while it is in the middle of a query (this behavior can be overridden by an administrative request to force the communications to be broken).

Transaction state

Each time the client connection changes transaction state, for example, by issuing an explicit `BEGIN TRAN`, `ROLLBACK TRAN`, or `COMMIT TRAN` statement, this state is monitored by the OpenSwitch thread. The thread attempts to protect the connection against administrative switch requests until the connection is no longer in an open transaction. This behavior can be overridden by an administrative request to force the transaction to be broken.

Client-side cursor state

All client-side cursor requests are monitored and managed by OpenSwitch so that, during failover, all client-side cursors can be restored on the secondary server. See “Working with client-side cursors” on page 39.

Managing switch requests

A switch or failover request can occur due to either an administrative request or a failed server.

Each thread manages its own outgoing connection and responds to these switch requests.

When a switch request is received, the thread behaves in the following manner:

- 1 If the current connection is completely idle (that is, not actively being used to communicate with a remote server, and not involved in an open transaction), the thread immediately switches the connection without any interruption of activity.
- 2 If the connection is not idle, and the switch request explicitly requests that the connection be switched immediately, or the switch request is due to a failed remote server, a deadlock message is issued to the client (indicating that the current transaction has been rolled back), and the connection is switched.
- 3 If the connection is not idle, and the switch request did not specify any limit on how long the switch is to take, the thread switches the connection as soon as the client finishes communicating with the remote server, and exits the top level transaction.

Using connection caching

Use the connection caching feature of OpenSwitch to improve the performance of applications that rapidly create and destroy connections while running. For example:

- Web applications – the majority of Web applications are built around small common gateway interface (CGI) programs or scripts that run independently of the actual Web server. Due to the transitory nature of CGI applications, these programs are implemented to hold a database connection only for the duration of the query issued by the CGI.
- Site handlers – a site handler refers to a special Adaptive Server connection that is created when a query is issued between two servers, such as:

```
exec SERVER2...sp_helpdb
```

When this type of query is issued, the source Adaptive Server creates a site handler to multiplex all future queries to SERVER2. Only one physical connection is maintained between the two servers and future queries are initiated faster.

However, when SERVER2 is an instance of OpenSwitch, even though only one physical connection is coming in, multiple outgoing connections must still be maintained by OpenSwitch, one for each query issued over the site handler. OpenSwitch drops its outgoing connection after each remote procedure call (RPC) that is issued over the site handler.

Defining connection pools and caching

When a client connection fails, the OpenSwitch default behavior is to immediately close down the outgoing connection to the remote Adaptive Server. However, for applications that rapidly create connections, issue a small query, then close the connection, the OpenSwitch default behavior can impose significant overhead.

To override the default OpenSwitch behavior, supply the optional *CACHE* value when you define a pool.

Note You create pools and specify *CACHE* values when you configure OpenSwitch, either using the GUI configuration tool during or after installation (see the OpenSwitch installation guide, Chapter 3, “Configuring OpenSwitch”), or by manually editing the OpenSwitch configuration file (see Chapter 4, “Using the Configuration File”).

The *CACHE* value indicates the number of seconds that an outgoing connection is maintained after a client application disconnects from an Adaptive Server.

For example:

```
[POOL=POOL_A:MODE=CHAINED,CACHE=30]
servers:
    ...
connections:
    type: site
```

This configuration example specifies that all users of the pool maintain cached copies of their outgoing connection for up to 30 seconds following a disconnection.

If the same client attempts to reconnect using the same user name and password, the connection is immediately reassigned to the user without creating a new outgoing connection. If 30 seconds has passed (the *CACHE* value) and no client has attempted a connection, the connection is dropped.

Changing a pool’s *CACHE* duration does not affect existing cached connections—it only affects the caching of future connections. To manage or change the cache value associated with a pool in a running OpenSwitch, use the `rp_pool_cache` registered procedure. Changing the caching time on a pool does not affect existing cached connections, it affects only the caching of future connections. See `rp_pool_cache` on page 152 for more information.

Restoring cached connections

When a user connects to OpenSwitch, the list of cached connections is searched, in an unspecified order, for the first connection owned by the same user name and password. If a cached connection is not found, a new connection is established as though caching were not enabled in the pool.

However, when a cached connection is found, the connection is re-established to the thread representing the newly connected user. The database context (transaction state) is then cleared to ensure that the new connection has a fresh context to work in, without any residual settings from the previous login session.

Uncached connections

When a client disconnects from an Adaptive Server, the state of the connection is evaluated to determine whether to cache the connection. OpenSwitch does not cache these connections:

- Connections with an open transaction
- Connections not established due to login failure
- Connections with Open Client-side cursors
- Connections where the maximum number of cache threads is reached, as specified by *CACHE_THREAD* option in the [CONFIG] section of the OpenSwitch configuration file

In all of these states, restoring the connection to a user would place the user's application in an uncertain state. To avoid this risk, OpenSwitch discards these connections when the client exits.

Caching and state

Because a cached connection never disconnects from the remote server, any state information held in that server regarding the user's session is maintained between the time the user disconnects from OpenSwitch and reconnects to the cached connection.

This means that temporary tables and "SET" options (except for "set database," which is cleared) are maintained even after a user disconnects. Evaluate carefully how you use connection caching.

Viewing cached connection details

When a user disconnects from a pool that has connection caching enabled, the query currently running on the connection is canceled, and the outgoing connection to the Adaptive Serve is stored in an internal list. These connections are not associated with any particular thread, and therefore, cannot be queried through `sp_who`.

To establish how many cached connections there are for a server, particular user, application, or host, execute `rp_who`. See `rp_who` on page 201 for syntax, instructions, and examples.

To determine how long a connection has been cached, execute `rp_dump` registered procedure, where all connections show up with a state of “CACHE.” For example:

```
<Cache thread: state=CACHE coord=<NONE>>
server mask=0x0, busy time='04/30/04 21:52:57',
transtate=CS_TRAN_UNDEFINED,
app='isql', user='sa', host='oswaix1', db='master',
conn=0x3114e368, current='monsoon_ASE1', next='monsoon_ASE1', pool='POOL1',
proc=0x0, cap set=CS_FALSE, next cursor=0, reason code=0,
reason text='', function=''
(return status = 0)
```

See `rp_dump` on page 137 for more information.

Setting connection caching properties, options, and database context

The connection’s server name, user name, and password are fixed when a cache is established. However, you can change other connection properties dynamically when the connection is opened.

Follow these guidelines to avoid problems with inconsistent connection state:

- Connection properties – connection properties affect client-side connection behavior.
- Server-side connections options – use the Client-Library `ct_options` routine to set language commands or equivalent ODBC calls. All options affect the server’s response to commands sent on the connection. See Chapter 3, “Routines,” in the *Open Client Client-Library/C Reference Manual* for details.
- Database contexts – different applications from the same user may use different databases. To avoid problems, explicitly specify the database within an applications when caching is turned on.
- Set options and properties that your code requires when the client application obtains a connection.
- If an application shares cached connections with other applications, set properties and options that have been changed back to the original values before releasing the connection.

- If an application is the only connection allowed for a cached pool, and no other components use the same pool, you do not need to set options and properties back to the original values. However, you should still set the database context every time the client application reconnects, even if the application is using cached connections.

Managing failures

This section discusses how failures are detected and how OpenSwitch responds to each type of failure.

Failure detection

OpenSwitch has a connection monitor (CMON) thread that monitors Adaptive Server and asynchronously notifies threads as soon as connectivity to the remote server is lost.

At the first successful connection attempt to a remote server, OpenSwitch starts a new CMON thread to monitor the state of the remote server. This thread runs as long as the remote server it is monitoring remains online. At the first indication of a remote server failure, the CMON thread notifies all client threads currently connected to the failed server that they (or a server specified by the coordination module) should connect to the next available server in the pool. The busy clients are issued a deadlock error (1205) message and switched immediately to the next server while the idle clients are switched when they attempt to execute their next commands. This failover detection process is always localized to a single connection, and connections are never automatically switched by OpenSwitch without either a switch request or a server down event. If you want all client connections to fail over to the secondary data server at the same time, you can use a coordination module to coordinate the various connections. See the *OpenSwitch Coordination Module Reference Manual* for complete instructions on creating and using coordination modules.

Deadlock messages

If the client connection is actively in the middle of communications with the remote server, or in the middle of a transaction (or a nested transaction), the thread notifies the client that the connection has been lost by issuing a deadlock message. For example:

```
Msg 1205, Level 13, State 0
```

```
Server 'OpenSwitch'
```

```
Your command (process id #8) was deadlocked with another process and has been  
chosen as deadlock victim. Re-run your command.
```

The message is used by an Adaptive Server to indicate that the current transaction has been aborted due to a resource contention issue, and should be restarted.

Note An Adaptive Server issues a return status of -3 to all client applications running a stored procedure when a deadlock message is issued. Because OpenSwitch cannot detect when a client is running a stored procedure from within a language batch, the return status of -3 is returned only to those clients that issued the RPC request through the Open Client RPC mechanism.

Working with client-side cursors

A cursor is a record pointer in Adaptive Server. The cursor points to the first record in the file when a database file is selected and the cursor is opened. Using various commands, you can move the cursor forward, backward, to the top of the file, the bottom of file, and so on.

A client-side cursor is a cursor declared through Open Client calls or Embedded SQL[™]. Open Client keeps track of the rows returned from Adaptive Server and buffers them for the application. Client-side cursors are similar to regular server-side cursors created with an explicit `declare cursor` command. Client-side cursors, however, are declared and controlled through special Open Client API calls in the client application, and Open Client itself manages portions of the cursor context information, whereas server-side cursors are managed only within Adaptive Server.

Because OpenSwitch manages client-side cursors for the client application, it can restore the state of a given cursor during failover.

The default behavior of most Sybase ODBC drivers is to use client-side cursors rather than issuing direct SQL against the database. Sybase strongly recommends that you read this section carefully before deciding to use client-side cursors.

Cursors within dynamic SQL

A cursor within dynamic SQL comes in two forms from the client:

- `ct_cursor`
- `ct_dynamic`

Each of these generates a different type of event notification; `SRV_CURSOR`, and `SRV_DYNAMIC` respectively.

Failover handling

During an automatic failover due to a remote server failure, or an explicit administrative switch request, OpenSwitch manages active client-side cursors in the following manner:

- 1 If an outgoing connection is still available to the remote server, the cursor is explicitly closed on the server.
- 2 If the client connection managing the cursor was involved in an open transaction at the time of the switch request or failure, a deadlock is issued to notify the client application that any updates or deletes performed upon the cursor have been rolled back.
- 3 After the outgoing connection to the failover remote server has been reestablished, all client-side cursors managed by OpenSwitch are re-created on the server.
- 4 After a client-side cursor has been re-created, it is repositioned on the same row as it was on the previous server.

OpenSwitch Sybase Failover support

Adaptive Server Enterprise 12.0 and later support Sybase Failover capability, which enables the secondary, or companion, server to take over when the primary server goes down. OpenSwitch works with Adaptive Server clusters and processes failover events.

Failover behavior with dynamic SQL

OpenSwitch maintains a list of any dynamic SQL commands that are prepared but not yet deallocated. If the list is not null, OpenSwitch resends the `DYNAMIC_PREPARE` statements for each command to the secondary server using statement ID and query content stored in the list for each of the threads. The clients are then switched over to the secondary server.

If a client connection is busy during failover, a 1205 error is issued back to the client. It is left to the client to reissue the command.

If any client was in a busy state at the time of failover, an error message is issued and the statement must be rerun.

If there are open cursors in the dynamic SQL statement, OpenSwitch redeclares the open cursors on the secondary server.

Cursor repositioning

As noted in step 4 in “Failover handling” on page 40, all cursors that are re-created due to a failover are automatically repositioned by OpenSwitch to the same row as the cursor that was sitting on the primary server. This has several implications concerning application performance and reliability.

To perform this repositioning, OpenSwitch tracks the number of rows that have been explicitly fetched by the client application. During the repositioning process, an identical number of fetches are performed against the secondary server to place the cursor on the appropriate row. Sybase recommends that client applications not use cursors to fetch unusually large result sets to avoid long delays during failover while client-side cursors are repositioned.

Repositioning is based only upon the number of rows returned; not upon row contents. Therefore, both the primary and secondary servers must be configured identically and must be synchronized with each other to avoid situations in which the cursor is restored upon an inappropriate row. In particular, avoid using cursors on partitioned tables without supplying an explicit ORDER BY clause in the cursor, otherwise, Adaptive Server cannot guarantee the order of rows returned during the cursor fetch.

Enabling SSL support

Secure Sockets Layer (SSL) is supported between clients and OpenSwitch. To enable SSL in OpenSwitch, you must:

- Have a *trusted.txt* file in *%SYBASE%\ini* on Windows and in *\$SYBASE/config* on UNIX.
- Specify the SSL security mechanism as a filter on the master and query lines in the *sql.ini* on Windows and in the *interfaces* file on UNIX.
- Define the SSL filter in the *libtcl.cfg* file; for example:

```
[FILTERS]
ssl=libsybfssl.so
```

See the SSL overview section in Chapter 2 of the *Open Client Client-Library/C Reference Manual* for details.

Starting and Stopping OpenSwitch and RCMs

This chapter describes how to start and stop OpenSwitch.

Topic	Page
Starting and stopping OpenSwitch on UNIX	43
Starting and stopping OpenSwitch on Windows	44
Using encrypted user names and passwords	48
Using command line options	49
Starting and stopping the RCM from OpenSwitch	54

Starting and stopping OpenSwitch on UNIX

OpenSwitch starts automatically after installation and configuration.

❖ Restarting OpenSwitch

1 In a terminal window, go to the `$SYBASE` directory.

2 In a C Shell, enter:

```
source SYBASE.csh
```

In a Bourne or Korn Shell, enter:

```
./SYBASE.csh
```

3 Go to `$SYBASE/OpenSwitch-15_0/bin`, and enter the following, where `-c` specifies the name of OpenSwitch configuration file to be used during start-up:

```
./OpenSwitch -c ../config/OpenSwitch.cfg
```

You can configure OpenSwitch to use encrypted user names and passwords for the user name and password entries that are in the OpenSwitch configuration file. See “Using encrypted user names and passwords” on page 48.

See “Using command line options” on page 49 for a list of options you can use to adjust the behavior of OpenSwitch.

❖ **Stopping OpenSwitch**

1 Connect to isql in administrative mode. In a terminal window, enter:

```
isql -U <admin_username> -P <admin_password> -S <server_name>
```

2 Enter:

```
rp_shutdown
```

Starting and stopping OpenSwitch on Windows

OpenSwitch starts automatically after you install and configure the product.

Note You cannot start OpenSwitch until you have configured the product. If you chose not to configure OpenSwitch during installation, see Chapter 4, “Using the Configuration File,” in this guide, and the *OpenSwitch Installation Guide*, Chapter 3, Configuring OpenSwitch.”

❖ **Starting OpenSwitch from the Start menu**

If you added OpenSwitch to the Windows Start menu (see Chapter 4, “Post Installation Tasks” in the *OpenSwitch Installation Guide*), use these instructions to start the product.

- Select Start | Programs | OpenSwitch | OpenSwitch.

❖ **Starting OpenSwitch from a command prompt**

1 Open a Command Prompt window, and go to the %SYBASE% directory.

2 In the same window, enter:

```
SYBASE.bat
```

3 At the command prompt, enter:

```
cd %SYBASE%\OpenSwitch-15_0\bin
```

4 In the same window, enter the following command, then press Enter the following, where -c is the name of the OpenSwitch configuration file to use during start-up:

```
start OpenSwitch.bat -c ..\config\OpenSwitch.cfg
```

You can configure OpenSwitch to use encrypted user names and passwords for the user name and password entries that are in the OpenSwitch configuration file. See “Using encrypted user names and passwords” on page 48.

❖ Stopping OpenSwitch from the command line on Windows

1 Open a Command Prompt window and go to %SYBASE%.

2 Enter:

```
.\SYBASE.bat
```

3 At the command prompt, log in to OpenSwitch as an administrator enter the following, where *<OpenSwitch_server_name>* is the name of the OpenSwitch server you want to stop:

```
isql -Usa -Psa -S<OpenSwitch_server_name>
```

4 Issue this command:

```
rp_shutdown
go
```

❖ Starting OpenSwitch as a Windows service

- 1 Go to %OPENSWITCH%\bin and use a text editor to set the *OPENSWITCH* and *SYBASE_OCS* variables to the correct path in the *OpenSwitch.bat* file. Save the file and close the text editor.
- 2 From %OPENSWITCH%\bin, open a Command Prompt window, enter the following, where *<OpenSwitch configuration file path>* is the location of the OpenSwitch configuration file:

```
.\OpenSwitch.bat -c <OpenSwitch configuration file path> -R install
```

For example:

```
D:\Sybase\OpenSwitch-15.0\config\OpenSwitch.cfg
```

- 3 Open the Registry Editor. Select Start | Run, enter *regedit* in the Open text field, then click OK.
- 4 In the Registry Editor window, go to *HKEY_LOCAL_MACHINE | SYSTEM | CurrentControlSet | Services* and click the OpenSwitch server name.
 - a Select Edit | New | Key, then enter “Parameters” as the key name.
 - b Click the new Parameters key, select Edit | New | String Value, then enter “Application” as the new string value.

- c Double-click Application. When the Edit String dialog box appears, enter the following text on one line in Value Data field:

```
<OpenSwitch installation directory>\bin\OpenSwitch.bat -c  
<OpenSwitch configuration file path> -l <OpenSwitch log file path>
```

Where:

- *<OpenSwitch installation directory>* – is the drive and directory in which OpenSwitch is installed; for example:
D:\Sybase\OpenSwitch-15_0
- *<OpenSwitch configuration file path>* – is the location of the OpenSwitch configuration file; for example:
D:\Sybase\OpenSwitch-15.0\config\OpenSwitch.cfg
- *<OpenSwitch log file path>* – is the file location of the OpenSwitch log file; for example:
D:\Sybase\OpenSwitch-15_0\bin\OSWServer.log

Note Use the name of your actual OpenSwitch configuration file and log file when entering this information.

Click OK.

- d In the left pane of the Registry Editor window, click your service name. In the right pane of the Registry Editor, double-click the “ImagePath” string and delete the text in the Edit String dialog box.

Enter the full path to the *srvany.exe* file. For example:

```
D:\engapps\NTResKit\srvany.exe
```

Click OK.

Note *srvany.exe* is installed on Windows operating systems as part of the Resource Kit. If *srvany.exe* is not on your machine, download the file as part of the Windows Resource Kit for your platform at the Microsoft Windows Resource Kit Web page at <http://www.microsoft.com/windows/reskits/default.asp>.

- e Select Registry | Exit to close the Registry Editor.
- 5 Select Start | Settings | Control Panel | Administrative Tools | Services, right-click OpenSwitch *<OpenSwitch server name>* in the Services right pane, then click Properties.

- 6 In the Properties dialog box:
 - a Select the Log On tab.
 - b Select Local System Account and Allow Service To Interact With Desktop.
 - c Select the General tab.
 - d In the Startup Type box, select Automatic.
If you do not want OpenSwitch to start automatically the next time the Windows machine is restarted, change the Startup Type to Manual.
 - e Click Apply, then click OK to close the Properties dialog box.
- 7 In the Services right pane, right-click the OpenSwitch service and select Start.
- 8 Close the Services window.

❖ **Stopping the OpenSwitch service on Windows**

- 1 Right-click an empty space on the Windows taskbar, then click Task Manager.
- 2 Select the Processes tab.
- 3 Click on the OpenSwitch process and select End Process.
- 4 Close the Windows Task Manager window.
- 5 If this fails because the process is being used by an application, select Start | Settings | Control Panel | Administrative Tools | Services.
- 6 In the Services window, right-click the OpenSwitch service and select Stop.
- 7 Close the Services window.

❖ **Removing OpenSwitch as a service**

- 1 Shut down OpenSwitch if it is running.
- 2 Open a Command Prompt window and go to %OPENSWITCH%\bin.
- 3 At a command prompt, enter the following, where *<OpenSwitch configuration file path>* is the location of the OpenSwitch configuration file:

```
.\OpenSwitch.bat -c <OpenSwitch configuration file path> -R remove
```

For example:

D:\Sybase\OpenSwitch-15.0\config\OpenSwitch.cfg

This stops the service and removes it from the Windows Registry.

Using encrypted user names and passwords

Configure OpenSwitch to use encrypted user names and passwords by using a text editor to modify the following parameters in the OpenSwitch configuration file:

- *ADMIN_USER*
- *ADMIN_PASSWORD*
- *COORD_USER*
- *COORD_PASSWORD*
- *CMON_USER*
- *CMON_PASSWORD*
- *COMPANION_ADMIN_USER*
- *COMPANION_ADMIN_PASSWORD*

If OpenSwitch is configured for user name and password encryption, all user names and passwords in the OpenSwitch configuration file as well as those in the coordination module must be encrypted. See “Manually editing configuration options” on page 63.

❖ **Encrypting user names and passwords in the configuration file**

- 1 Shut down OpenSwitch using `rp_shutdown` (see `rp_shutdown` on page 190).
- 2 Restart OpenSwitch with the `-E` flag (see “Using command line options” on page 49).
- 3 OpenSwitch prompts for each user name and password in the configuration file. Once all of the entries are made, OpenSwitch writes the encrypted user names and passwords to the console.

You can use a file name as an optional argument with the `-E` flag so the encrypted user names and passwords are written to the specified file as well as the console. If an argument is not given, OpenSwitch writes the information only to the console. This is an example of the output:

```
ADMIN_USER =<encrypted username>
ADMIN_PASSWORD = <encrypted password>
COORD_USER = <encrypted username>
COORD_PASSWORD = <encrypted password>
CMON_USER = <encrypted username>
CMON_PASSWORD = <encrypted password>
COMPANION_ADMIN_USER = <encrypted username>
COMPANION_ADMIN_PASSWORD =<encrypted password>
CMON_USER = <encrypted username>
CMON_PASSWORD = <encrypted password>
SERVER_NAME = <server name in plain text>
USERNAME_PASSWORD_ENCRYPTED = 1
```

Note In this example, two *CMON_USER* and *CMON_PASSWORD* entries display. The first entry applies to all the servers for which the user has not explicitly set the *CMON_USER* and *CMON_PASSWORD*. The second *CMON_USER* and *CMON_PASSWORD* was explicitly set by the user for a specific server.

- 4 With a text editor, modify the configuration file to replace the non-encrypted values with the new encrypted values. Verify the *USERNAME_PASSWORD_ENCRYPTED* option is set to 1.
- 5 Restart OpenSwitch.

To encrypt a user name and password that are not in the OpenSwitch configuration file, such as *ping_user* and *ping_password*, which are in the *cm1.c* sample, start OpenSwitch with *-p<username>* or *-p<password>*. See “Using command line options” on page 49. The encrypted string displays on the console, which you can then cut and paste to where it is needed.

Using command line options

Use the command line options described in this section to adjust the behavior of OpenSwitch.

Use the configuration GUI to set these options during or after installation; see Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide*. You can also set or change these options using a text editor to manually modify the OpenSwitch configuration file. See Chapter 4, “Using the Configuration File.”

- **-a *charset*** – sets the name of the default character set used during communications between the client connections and OpenSwitch. You can also set this option using the *CHARSET* configuration option.
- **-c *conflictive*** – specifies the name of a configuration file to read during start-up. If supplied, this option must be the first option on the command line. All other occurrences of **-c** are ignored. Portions of the configuration file contents are overridden by any subsequent command line options that you supply.
- **-C *conn_flags*** – when running the symbolic (debug) version of OpenSwitch (located in *\$OPENSWITCH/devbin* on UNIX and in *%OPENSWITCH%\devbin* on Windows) and linking with the Sybase instrumented libraries (located in *\$SYBASE/OCS-15_0/devlib* on UNIX and in *%OPENSWITCH%\devlib* on Windows), you can supply one or more of the flags described below to turn on connection-level debugging.

Value	Description
a	All debugging messages
d	Diagnostic messages
p	Networking protocols
s	Protocol state

- **-d *dbg_log*** – when used in conjunction with **-C** and **-X**, all debugging output is redirected to the file *dbg_log*, rather than the default *OpenSwitch.log*.

This option is valid only when running the symbolic (debug) version of OpenSwitch (located in *\$OPENSWITCH/devbin* on UNIX and in *%OPENSWITCH%\devbin* on Windows) and linking with the Sybase instrumented libraries (located in *\$SYBASE/OCS-15_0/devlib* on UNIX and in *%OPENSWITCH%\devlib* on Windows).

- **-e** – enables echoing of all message log information is sent to *stderr* while OpenSwitch is running (all log messages go directly to the *log_file*).
- **-E *<filename>*** – use when encryption is required for the user names and passwords in the OpenSwitch configuration file. By default, **-E** sends its output to *stderr*. If the optional file name is given, the output is also sent to the specified file. When the **-E** flag is used, OpenSwitch prompts you for values for each of the following options in the [CONFIG] section:
 - *ADMIN_USER*
 - *ADMIN_PASSWORD*

- *COORD_USER*
- *COORD_PASSWORD*
- *CMON_USER*
- *CMON_PASSWORD*

OpenSwitch allows up to 29 cleartext characters to be entered for each request.

- -f – enables full pass-through mode for language commands and RPC commands. This mode may improve performance, but disables the server from tracking database context and transaction states during the switching process.

Note For information about configuring “full pass-through mode” using the *FULL_PASSTHRU* OpenSwitch configuration parameter, see “[CONFIG]” on page 64.

- -h – prints usage message.
- -I *interfaces* – indicates that OpenSwitch should start using the specified *interfaces* (UNIX) or *sql.ini* (Windows) file rather than the default of *\$SYBASE/interfaces* (UNIX) or *%SYBASE%\ini\sql.ini* (Windows). This option is an uppercase “I”.
- -l *log_file* – sends all OpenSwitch output to *log_file* rather than the default file of *OpenSwitch.log*. (This option is a lowercase “l”.)
- -n *server_name* – specifies the name of the OpenSwitch in the *interfaces* file; or the Windows *sql.ini* file. If not supplied, *server_name* defaults to *OPENSITCH*.
- -O – when you have *MUTUAL_AWARE* set to 1, which enables mutually-aware OpenSwitch servers, using -O overrides the data in the Adaptive Server configuration tables with the information from the OpenSwitch configuration file.
- -p – allows you to encrypt a particular string. The output is written only to the console. This is useful for applications (such as coordination module applications) where an encrypted user name or password is needed and it is a user name or password that is not processed with the -E option.
- -r – enables the resource monitor thread.

- `-s srv_flags` – when OpenSwitch is started using the Sybase instrumented libraries (located in `$$SYBASE/OCS-15_0/devlib` on UNIX and in `%SYBASE%\OCS-15_0\devlib` on Windows), you can use one or more of the flags described in the following table to enable Open Server-level debugging messages.

Value	Displays
a	TDS attention packets
d	TDS data information
e	Server events
h	TDS header information
m	Message queue usage
n	Network driver information
p	Network driver parameter information
q	Run queue information
r	Network driver data information
s	Network driver memory information
t	TDS tokens
w	TCL wake-up request

- `-S stacksize` – specifies the stack size for the OpenSwitch server. This overwrites the `STACKSIZE` setting in the configuration file.
- `-t dbg_flags` – enables OpenSwitch specific debugging messages. Use one or more of the options described below for `dbg_flags`.

Note You can turn these flags on and off at runtime using `rp_debug`.

Value	Description
a	Enables all possible debugging flags.
b	Displays attempts to set or test configuration options as described in the configuration file.
c	Displays information about result handling of client-side cursors.
C	Logs interactions between a mutually-aware OpenSwitch, its companion OpenSwitch, and Adaptive Servers.
d	Logs access to data items attached to each thread's user data.
D	Displays information about the handling of dynamic SQL statements.
e	Logs all error messages passing through the OpenSwitch error handlers, even those that are normally suppressed.

Value	Description
f	Shows connection progress information when OpenSwitch is interacting with the coordination module.
F	Display messages related to a coordination module (CM).
g	Displays operations involving security negotiations.
h	Displays messages when entering each event handler.
i	Displays progress information concerning the switching process during a call to <code>rp_switch</code> , such as success or failure of each switch, and which connections fail to go idle within the specified period of time.
j	Shows the connection caching activity.
k	Displays activity of the timer thread (the thread that is responsible for calling timed callbacks within OpenSwitch).
l	Dumps every SQL statement issued through the <code>SRV_LANGUAGE</code> event handler to <i>log_file</i> .
m	Displays every memory allocation and de-allocation (more extensive information may be available at compile time).
n	Displays receipt and handling of cancel or attention requests from client connections.
o	Displays a message each time a command line option value is set or tested.
p	Displays manipulation, use, and assignments of server pools.
q	Displays information about the connection monitor activity.
r	Displays current state and actions of the internal resource monitoring thread.
R	Logs interactions between an OpenSwitch and replication coordination modules (RCMs).
s	Shows access and release of shared and exclusive internal locks (used to prevent concurrent access to internal data structures).
S	Displays the SQL statement that is being executed as part of <code>rp_replay</code> calls.
t	Displays activities of the timer thread that is responsible for periodically waking other sleeping threads.
u	Displays information about result sets being returned to client threads.
U	Logs the user action, such as <code>CUSTOM</code> or <code>MANUAL</code> script execution during a companion OpenSwitch or Adaptive Server failure.
v	Logs ping operations and responses from remote machines.
x	Displays mutex accesses (more detailed view on shared locks).

- **-T** – truncates *log_file* at start-up rather than appending to the end of an existing file.
- **-u *nusers*** – sets the maximum number of client connections allowed (default is 1000).
- **-v** – displays the OpenSwitch version numbers to *stderr*, then exits.
- **-X *ctx_flags*** – when running the symbolic (debug) version of OpenSwitch and linking with the Sybase instrumented libraries located in *\$SYBASE/OCS-15_0/devlib* on UNIX and in *%SYBASE%\OCS-15_0\devlib* on Windows, you can supply one or more of the flags listing in the following table to turn on context level debugging in the server.

Value	Description
s	API state messages
e	API error messages
m	Memory allocation messages
n	Network accesses

Starting and stopping the RCM from OpenSwitch

OpenSwitch version 15.0 and later allows the replication coordination module (RCM) to automatically start and stop when you start and stop OpenSwitch. This functionality is supported by:

- Parameters *RCM_AUTOSTART*, *RCM_RETRIES*, *RCM_PATH*, *RCM_CFG_FILE*, *RCM_LOG_FILE*, and *RCM_SECONDARY* in the [CONFIG] section of the OpenSwitch configuration file.
- Registered procedures *rp_rcm_startup*, *rp_rcm_shutdown*, *rp_rcm_connect_primary*, and *rp_rcm_list*. See “Registered Procedures” on page 125 for syntax and use instructions.
- *RCMNAME* parameter in the RCM configuration file.

Requirements

See the *OpenSwitch Coordination Module Reference Manual*, Chapter 4, “Using the Replication Coordination Module,” for requirements and instructions on implementing a redundant high-availability, warm-standby environment.

Configuring an RCM to start automatically from OpenSwitch

When you configure OpenSwitch using the configuration GUI during or after installation, and select replication coordination module on the second configuration screen, the product is automatically configured to start and stop the RCM when you start and stop OpenSwitch.

If you did not select replication coordination module in your initial OpenSwitch configuration, you must re-run the configuration GUI, make this selection, and complete the associated options on the various screens that display. Use the instructions in this section to complete these tasks.

❖ Configuring an RCM autostart

- 1 Shut down OpenSwitch if it is running.
- 2 Start the OpenSwitch configuration GUI.

Note You must configure OpenSwitch to use an RCM by selecting Replication Coordination Module on the second screen of the OpenSwitch configuration GUI.

If you chose not to configure OpenSwitch during installation, start the configuration tool and use the instructions in “Configuring OpenSwitch using the GUI” in Chapter 2 of the *OpenSwitch Installation Guide* to complete the configuration.

- 3 After you complete the configuration and close the configuration GUI, use a text editor to open the OpenSwitch configuration file.

The configuration file is located at `%OPENSWITCH%\config\<cfg_file>` on Windows and at `$OPENSWITCH/config/<cfg_file>` on UNIX, where `<cfg_file>` is the name of your configuration file, for example, *OpenSwitch.cfg*.

- 4 Edit the following values in the file:

Parameter	Description
<i>RCM_AUTOSTART</i>	<p>Instructs OpenSwitch whether to start the RCM.</p> <p>Enter:</p> <ul style="list-style-type: none"> 0 – zero (0) for false. If you do not select Replication Coordination Module, this value defaults to 0. 1 – for true, which autostarts the RCM. When you choose Replication Coordination Module on the second screen of the configuration GUI, the value for this parameter is set to 1.
<i>RCM_RETRIES</i>	<p>Tells OpenSwitch how many times to retry starting the RCM. If the RCM fails for reasons other than the user requesting that the RCM be shut down, OpenSwitch attempts to restart the RCM. If an unrequested shutdown of the RCM occurs within one minute of starting, OpenSwitch logs an error and does not attempt to restart the RCM.</p> <p>Enter:</p> <ul style="list-style-type: none"> 0 – zero (0). OpenSwitch does not attempt to restart the RCM. If you do not select Replication Coordination Module, this value defaults to zero (0). 1 to . . . – any numeric value that represents the number of times OpenSwitch should attempt to restart the RCM. <p>This parameter is set to 1 when you choose Replication Coordination Module.</p>
<i>RCM_PATH</i>	<p>Enter the location where OpenSwitch should look for the RCM executable. If you do not enter a value, OpenSwitch runs the RCM located in <i>\$OPENSWITCH/bin</i> on UNIX systems or in <i>%OPENSWITCH%\bin</i> on Windows systems; where <i>OPENSWITCH</i> is the installation directory.</p> <p>This parameter has a null value if you do not specify a path.</p>
<i>RCM_CFG_FILE</i>	<p>Enter the file and directory path where the RCM configuration file is located. This parameter has a null value if you do not specify a path.</p>
<i>RCM_LOG_FILE</i>	<p>Enter the file and directory path where the RCM log file should be created. This parameter has a null value if you do not specify a path.</p>
<i>RCM_SECONDARY</i>	<p>Instructs OpenSwitch whether the RCM that is being started is a primary or a secondary RCM.</p> <p>Enter:</p> <ul style="list-style-type: none"> 0 – zero for a primary RCM. This is the default value. 1 – for a secondary RCM.

See “OpenSwitch sample configuration file” on page 57 for sample values.

- 5 Save the configuration file, close the text editor and restart OpenSwitch.
- 6 In a text editor, open the RCM configuration file located in *%OPENSWITCH%\config* on Windows and in *\$OPENSWITCH/config* on UNIX.

- 7 Edit the *RCMNAME* configuration parameter. To enter a unique name that allows OpenSwitch to identify the RCMs to which it is attached. OpenSwitch maintains an internal list of registered RCMs and uses the list to identify RCMs to shut down, or to display to the client application when *rp_rcm_list* is issued.
- 8 Save the file and close the text editor.
- 9 Optionally, use these RCM registered procedures to work with this feature: *rp_rcm_startup*, *rp_rcm_shutdown*, *rp_rcm_connect_primary*, *rp_rcm_list*. See “Registered Procedures” on page 125 for syntax and use instructions

OpenSwitch sample configuration file

The following example displays the parameters used in the [CONFIG] section of a sample OpenSwitch configuration file that are applicable to automatically starting the RCM when OpenSwitch starts.

```
[CONFIG]
SERVER_NAME      = rcm_osw
...
...
...
RCM_AUTOSTART    = 1
RCM_RETRIES      = 3
RCM_PATH         = /OpenSwitch-15_0/bin/rcm
RCM_CFG_FILE     = /OpenSwitch-15_0/config/rcm1.cfg
RCM_LOG_FILE     = /OpenSwitch-15_0/logs/rcm.err
RCM_SECONDARY    = 0.....
```

RCM sample configuration file example

Shows the *RCMNAME* parameter used in a sample RCM configuration file.

```
LANGUAGE = us_english
CHARSET = iso_1
OPENSWITCH = rcm_osw
SECONDARY_OPENSWITCH = rcm_osw2
COORD_USER = switch_coord
COORD_PASSWORD = switch_coord

RCMNAME = rcm1

REP_SERVER = osw_rs
```

```
RS_USER = sa
RS_PASSWORD =

ACTIVE_ASE = active_ase
ACTIVE_USER = burt
ACTIVE_PASSWORD = burt_pwd
ACTIVE_DBS = pubs2

STANDBY_ASE = standby_ase
STANDBY_USER = alice
STANDBY_PASSWORD = alice_pwd
STANDBY_DBS = pubs2

LOGICAL_CONN = LDS.LDB

REQUIRED_DBS = pubs2

APP_POOL = Application

FAILOVER_WAIT = 30
MONITOR_WAIT = 30

RS_FAILOVER_MODE = SWITCH
SWITCH_USERS = 1
```

Using the Configuration File

This chapter describes the contents of the OpenSwitch configuration file, which you can specify when you start OpenSwitch.

Topic	Page
Introduction	59
Editing the OpenSwitch configuration file	60
Creating or editing a configuration file	62
Manually editing configuration options	63

Introduction

When you install OpenSwitch, you can use the graphical configuration tool to configure the product. When you finish the configuration, OpenSwitch creates a configuration file that stores your selected configuration settings. If you choose not to use the configuration tool during or after installation, you must manually create a configuration file with the settings appropriate to your installation.

You may want to manually create or edit the OpenSwitch configuration file to:

- Use an existing configuration file from an earlier version of OpenSwitch and add parameters from the version of OpenSwitch to which you have upgraded.
- Configure the OpenSwitch manually.
- Manually change configuration values after the initial OpenSwitch configuration.

This chapter contains instructions on creating a new configuration file, formatting tips for editing a new or existing configuration file, and a list of all configuration file parameters for the current version of OpenSwitch.

You can also use the details provided for configuration parameters as a reference when using the configuration tool.

Note If you manually edit the OpenSwitch configuration file to initially configure OpenSwitch, or to change configuration values after initial configuration, you may also need to edit the *sql.ini* (Windows) or *interfaces* (UNIX) file. See the *OpenSwitch Installation Guide*, Chapter 3, “Configuring OpenSwitch” for more information.

Editing the OpenSwitch configuration file

The OpenSwitch configuration file consists of multiple sections, each of which is identified by the name of the section enclosed in square brackets. For example:

```
[SECTION_NAME]
```

No section is required. Empty lines, or lines that begin with a pound sign (#) are ignored by OpenSwitch.

You can specify options in some sections like this:

```
[SECTION_NAME:OPTION1=VALUE,OPTION2]
```

Set true and false, or on and off options using zero (0) to indicate off or false, and one (1) to indicate on or true.

Using wildcards

OpenSwitch allows you to use wildcard expressions in its configuration file to represent values for the connection attributes.

If you use a wildcard expression to define an attribute under the [POOL] connections section, for example:

```
[POOL=POOL1:MODE=CHAINED,CACHE=0]
connections:
username=[abcde] %
```

Then all users whose user names start with a letter from a to e are channeled to POOL1.

Or, if you use a wildcard expression to define an attribute under the [LIMIT_RESOURCE] section, for example:

```
[LIMIT_RESOURCE:ACTION=KILL,BUSY=100]
  appname: [ei]sql
```

Then an application with the name of “esql” or “isql” is governed by the resource monitor and killed if it spends longer than 100 seconds in a transaction.

OpenSwitch wildcard expressions are identical to those used in Adaptive Server Enterprise, except for the escape expression.

Brackets ([])

Use square brackets ([]) to enclose a range of characters, such as “[a-f]”, or a set of characters, such as “[85Rza].” When ranges are used such as [char1-char2], all values in the ASCII range between *char1* and *char2* match.

For example, if you use 8-bit ASCII, “[0-z]” matches 0–9, A–Z, and a–z, as well as several punctuation characters.

The wildcard expression: [dD]og matches these strings:

```
dog
Dog
```

The wildcard expression [A-Z]at matches these strings:

```
Pat
Hat
Cat
```

Caret (^)

When included as the first character within a set of brackets, you can use the caret symbol (^) to negate matching a set of characters within the brackets.

For example, the expression: “[^A-Z]at” matches these strings:

```
cat
hat
pat
```

It does not, however, match:

```
Cat
Hat
Pat
```

Escape (\)

Use the escape (\) wildcard to negate special meaning of another wildcard character. For example, to include the “%” as a literal part of an expression, “90\%” matches the string 90%.

You can use the escape wildcard to negate the meaning of any character within a wildcard expression.

Percent (%)

The percent (%) wildcard represents a string of zero or more characters. For example, the wildcard expression “sco%” matches these strings:

```
scott
scooter
scope
```

But it does not match:

```
escort
```

Underscore (_)

The underscore (_) wildcard matches any single character. For example, the expression “_op” matches these strings:

```
pop
mop
top
```

But it does not match:

```
stop
```

Creating or editing a configuration file

Use the procedures in this section to create a new OpenSwitch configuration file or edit an existing one.

❖ Setting up the OpenSwitch configuration file

- 1 To create a new OpenSwitch configuration file, go to `%OPENSWITCH%\config` on Windows or `$OPENSWITCH/config` on UNIX and copy the *sample.cfg* file by entering the following, where *<cfg_file>* is the name of your server configuration:

```
cp sample.cfg <cfg_file>
```

For example, *OpenSwitch.cfg*.

Use a text editor to set the values in the configuration file you just created. Use the information in the tables in “Manually editing configuration options” on page 63 for the definitions of the values to provide.

Note You can also run the standalone configuration GUI to set the configuration parameters. See Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide*.

- 2 Save the file and close the text editor.
- 3 Go to `%OPENSWITCH%` and create a `logs` directory:

```
mkdir logs
```
- 4 Start OpenSwitch. You cannot start OpenSwitch until you have configured the product.

❖ **Editing an existing OpenSwitch configuration file**

- 1 Stop OpenSwitch if it is running.
- 2 Go to go to `%OPENSWITCH%\config` on Windows or `$OPENSWITCH/config` on UNIX.
- 3 In a text editor, open the OpenSwitch configuration file; for example, *OpenSwitch.cfg*.
- 4 Modify the files as necessary using the remainder of this chapter for reference.
- 5 Save the updated file and close the text editor.
- 6 Restart OpenSwitch.

Manually editing configuration options

The configuration file contains these sections:

- **[CONFIG]** – OpenSwitch server configuration values, which you can also set individually at a command line.
- **[SERVER]** – defines the settings and status of Adaptive Servers available for use within OpenSwitch.
- **[POOL]** – defines a group of Adaptive Servers and the set of connections for each group.

- [LIMIT_RESOURCE] – if you enable the OpenSwitch resource limiting feature at start-up with the -r flag or the *RMON* configuration option, use this section to specify the connections that are candidates for resource governing and which resources to limit.
- [COMPANION] – name of the OpenSwitch companion, and the administrator user name and password used to make a connection.

[CONFIG]

Description	Allows you to set OpenSwitch command line options. The [CONFIG] section can contain zero or more <i>NAME</i> options.
Format	<pre>[CONFIG] NAME = VALUE NAME = VALUE NAME = VALUE</pre>
Parameters	<ul style="list-style-type: none">• <i>NAME</i> and <i>VALUE</i> – configuration options specific to OpenSwitch, which are entered in the format: <pre>NAME=VALUE</pre><p>Where:</p><ul style="list-style-type: none">• <i>NAME</i> – is the option being set• <i>VALUE</i> – is the option’s setting<p>Table 4-1 on page 65 describes the options you can set (<i>NAME</i>), and the setting available for each option’s <i>VALUE</i>.</p><hr/><p>Note The options shown in Table 4-1 are listed alphabetically for easy reference. This is not the order in which the options appear in the configuration file.</p><hr/><p>In the table, the <i>VALUE</i> description includes the command line option, being set with this parameter. When you start OpenSwitch at the command line, you can enter the command line option to achieve the same results. For more information, see “Using command line options” on page 49.</p>

The *VALUE* description also indicates whether an option is configured dynamically or statically. A dynamic option indicates a newly configured value that takes effect immediately and affects all future connections; existing connections are not affected. Dynamic options usually affect individual connections. Static options cannot be changed by the user while OpenSwitch is running. You must stop and restart OpenSwitch before the changes take effect. Static options usually define the overall characteristics of the OpenSwitch server and its start-up options.

Note A value of DEFAULT sets the default values for any server not explicitly listed in a this section.

Table 4-1: [CONFIG] NAME and VALUE options

NAME	VALUE
<i>ADMIN_PASSWORD</i>	<p>This option has no command line equivalent.</p> <p>Enter the administrative user password. If <i>USERNAME_PASSWORD_ENCRYPTED</i> is set to 1, this option should contain the encrypted string of the administrative user password. To encrypt a password, use the -E or -p command line option. See “Using encrypted user names and passwords” on page 48.</p> <p>This option is configured dynamically.</p>
<i>ADMIN_USER</i>	<p>This option has no command line equivalent.</p> <p>Enter the name of the incoming user connections that should be considered administrative users. An administrative user has no outgoing connection to the remote Adaptive Server and is intended to perform only administrative tasks, mostly through registered procedure calls (RPCs).</p> <p>If the <i>USERNAME_PASSWORD_ENCRYPTED</i> option is set to 1, this option should contain the encrypted string of the administrative user name. To encrypt a user name, use the -E or -p command line options.</p> <p>This option is configured dynamically.</p>

NAME	VALUE
<i>API_CHECK</i>	<p>There is no command line equivalent for this option.</p> <p>This option indicates whether to enable or disable the validation of Server-Library arguments and state checking, and may be useful for debugging.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to check all APIs internally for invalid parameters before execution. This is the default. • 0 – to execute all internal APIs without checking for invalid parameters. This setting can speed up performance, but it should be used with caution. <p>This option is configured statically.</p> <hr/> <p>Warning! Do not set <i>API_CHECK</i> to zero (FALSE) unless your application has been completely debugged with the default setting of 1 (TRUE). If <i>API_CHECK</i> is 0 (zero), arguments are not validated before they are used. If OpenSwitch passes invalid arguments to Open Client or Client-Library, then OpenSwitch does not work correctly, resulting in memory corruption, memory access violations (UNIX core dumps), or incorrect results. No error messages are generated warning of the condition.</p>
<i>BCP_LOGGED</i>	<p>There is no command line equivalent for this option.</p> <p>Indicates whether the OpenSwitch supports bcp in operations. If a bcp operation is not logged and a failure occurs while the operation is executing, OpenSwitch cannot guarantee the recovery of all bcp committed transactions after the failover. Therefore, OpenSwitch allows only bcp in operations if the administrator explicitly confirms that the bcp operations are being logged, and are therefore recoverable through the Replication Server.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to specify that bcp operations are being logged and are recoverable through the Replication Server. • 0 – to specify that bcp operations are not being logged and are not allowed to go through OpenSwitch. This is the default setting. <p>This option is configured dynamically.</p>
<i>CACHE_THREADS</i>	<p>There is no command line equivalent for this option.</p> <p>Enter an integer value that represents the maximum number of connection threads saved in the OpenSwitch server connection cache.</p> <p>This option is configured dynamically.</p>
<i>CHARSET</i>	<p>Sets the -a command line option.</p> <p>Specifies the default character set used by OpenSwitch. This character set is used in communications with client connections.</p> <p>This option is configured statically.</p>

NAME	VALUE
<i>CMON</i>	<p>There is no command line equivalent for this option.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to use a single monitoring thread to monitor the health of each of the Adaptive Servers connected to OpenSwitch and to facilitate a failover when necessary. This must be set for OpenSwitch to support Adaptive Server HA clusters, but is also recommended for other non-cluster servers. This is the default. • 0 – to have each thread rely on receiving its own asynchronous notification from the Adaptive Server to tell it whether a failover is necessary. This setting can be used if there are no Adaptive Server HA clusters among the servers being connected to. <p>This option is configured statically.</p>
<i>CMON_FAIL_ACTION</i> Note Required only when <i>MUTUAL_AWARE=1</i> .	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is read-only when the <i>CMON</i> configuration parameter is set to 1 (the recommended setting), and when the CMON thread that monitors the health of the Adaptive Server fails to start. See Table 4-1 on page 65 for information about the <i>CMON</i> parameter.</p> <p>See “<i>CMON_FAIL_ACTION</i>” on page 119 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>
<i>CMON_PASSWORD</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the password for <i>CMON_USER</i>. The default is (empty string).</p> <p>If the <i>USERNAME_PASSWORD_ENCRYPTED</i> option is set to 1, this should contain the encrypted string of the CMON user password. To encrypt a password, use the -E or -p command line options. See “Using encrypted user names and passwords” on page 48.</p> <p>This option is configured statically.</p>

NAME	VALUE
<i>CMON_USER</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the user login used by the connection monitor (CMON) thread to connect to the back-end server. This must be an existing, valid login on each of the Adaptive Servers being used. Verify this user has basic privileges and can issue a “wait for delay” query to the remote data server.</p> <p>If the <i>CMON_USER</i> is not a valid Adaptive Server user, the client can be left in an undefined state when OpenSwitch is configured for failover mode and a failover occurs to the secondary Adaptive Server.</p> <p>Be sure to specify a valid CMON user name and that the user is a valid Adaptive Server user. For <i>CMON_WAITFOR_DELAY</i> to work properly, this user should not be an administrative user on the remote data server.</p> <p>If <i>USERNAME_PASSWORD_ENCRYPTED</i> is set to 1, this should contain the encrypted string of the CMON user name. To encrypt a user name, use the -E or -p command line options.</p> <p>This option is configured statically.</p>
<i>CMON_WAITFOR_DELAY</i>	<p>There is no command line equivalent for this option.</p> <p>Allows the user to configure the amount of time the CMON thread delays before the WAITFOR expires when it issues the “wait for delay” query.</p> <p>Enter an integer representing seconds. The default is 3600 (1 hour). Set the <i>CMON_WAITFOR_DELAY</i> to a lesser value if you want the shutdown to occur more quickly.</p> <p>Use this when you want the graceful shutdown of an Adaptive Server rather than a shutdown with no wait.</p> <p>This option is configured dynamically.</p>
<i>CMP_FAIL_ACTION</i>	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when network connectivity is lost to the companion OpenSwitch in a mutually-aware setup. Once the network is restored and the connection to the companion is re-established, the two OpenSwitch servers synchronize their configurations.</p> <p>See “<i>CMP_FAIL_ACTION</i>” on page 120 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>

Note Required only when
MUTUAL_AWARE=1.

NAME	VALUE
<i>CON_TRACE</i>	<p>Sets the -C command line option.</p> <p>Set connection-level tracing flags. See “Using command line options” on page 49 for the available settings. The output of <i>CON_TRACE</i> is directed to the file specified by the <i>DEBUG_FILE</i>.</p> <p>This option is configured dynamically.</p>
<i>CONNECTIONS</i>	<p>Sets the -u command line option.</p> <p>Enter the maximum number of user or threads the server can handle.</p> <hr/> <p>Note Set this parameter to a value smaller than the maximum number of file descriptors (see UNIX <code>ulimit -Ha</code> command) to prevent OpenSwitch from using more descriptors than are available.</p> <hr/> <p>This option is configured statically.</p>
<i>COORD_MODE</i>	<p>There is no command line equivalent for this option.</p> <p>Determines how OpenSwitch behaves when a coordination module (CM or RCM) is used. Enter:</p> <ul style="list-style-type: none"> • NONE – to not use a coordination module. This still allows coordination modules to be connected, but the modules do not receive any OpenSwitch notifications. • AVAIL – to use a coordination module if one is available. If a coordination module is not available, OpenSwitch decides how to deal with client connections. • ALWAYS – to use a coordination module, which must be available. If none is attached, all user connections requiring the services of a coordination module are refused until one becomes available. <hr/> <p>Note To use the RCM, you must set <i>COORD_MODE</i> to ALWAYS. The RCM can then coordinate the switch of users between the active and the standby Adaptive Servers so the OpenSwitch server does not allow users to connect unless the RCM is available. See the <i>OpenSwitch Coordination Module Reference Manual</i> for details.</p> <hr/> <ul style="list-style-type: none"> • ENFORCED – to use a coordination module, which must be available. If none is attached, all user connections requiring the services of a coordination module are refused with an informational message. <p>This option is configured dynamically.</p>

NAME	VALUE
<i>COORD_PASSWORD</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the password used by the coordination module (CM or RCM) logging in as <i>COORD_USER</i>. If the password is not correctly supplied, the coordination module is treated like any other OpenSwitch user and attempts to establish an outgoing server connection.</p> <p>If <i>USERNAME_PASSWORD_ENCRYPTED</i> is set to 1, this option should contain the encrypted string of the coordination module (CM or RCM) user password. To encrypt a password, use the -E or -p command line option. See “Using encrypted user names and passwords” on page 48.</p> <p>This option is configured dynamically.</p>
<i>COORD_TIMEOUT</i>	<p>The maximum amount of time (in seconds) it should take for a coordination module to respond to a notification before OpenSwitch makes the next CM ID active. When this parameter is set to zero (0), the default, the concurrent CM feature is disabled. See the <i>OpenSwitch Coordination Module Reference Manual</i> for more information about using concurrent coordination modules.</p> <hr/> <p>Note If you are using an RCM, <i>COORD_TIMEOUT</i> must be set to zero (0) for the RCM to start.</p>
<i>COORD_USER</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the user name used by coordination modules (CM or RCM) to connect to OpenSwitch. This user does not have an outgoing server connection established with it, and can register to receive coordination notifications.</p> <p>If <i>USERNAME_PASSWORD_ENCRYPTED</i> is set to 1, this option should contain the encrypted string of the coordination module (CM or RCM) user name. To encrypt a user name, use the -E or -p command line option. See “Using encrypted user names and passwords” on page 48.</p> <p>This option is configured dynamically.</p>
<i>CTX_TRACE</i>	<p>Sets the -X command line option.</p> <p>Enter context-level tracing flags. See “Using command line options” on page 49 for the available settings. The output of the CTX_TRACE is directed to the file specified by <i>DEBUG_FILE</i>.</p> <p>This option is configured statically.</p>
<i>CURSOR_PREREAD</i>	<p>There is no command line equivalent for this option.</p> <p>Enter an integer value that represents the number of rows to be returned with a single fetch request for a cursor.</p> <p>This option is configured dynamically.</p>

NAME	VALUE
<i>CUSTOM_SCRIPT</i>	<p>The path to the user-created script to invoke.</p> <hr/> <p>Note When <code>MUTUAL_AWARE=1</code>, the scripts for both OpenSwitch companions must perform the same action. When a server fails over and <code>SVR_FAIL_ACTION</code> is set to <code>MANUAL</code> or <code>CUSTOM</code>, only one of the companions executes the script that notifies the administrator or restarts the server.</p> <hr/> <p>See “Invoking custom and manual scripts” on page 111 for details about which exit codes to use in custom scripts.</p>
<i>DEBUG</i>	<p>Sets the <code>-t</code> command line option.</p> <p>Enter OpenSwitch debugging flags. See “Using command line options” on page 49 for valid flags to set.</p> <p>This option is configured dynamically.</p>
<i>DEBUG_FILE</i>	<p>Sets the <code>-d</code> command line option.</p> <p>Enter the path to and name of the file in which to place debugging output. This option is used only for the context <code>CTX_TRACE</code> and connection <code>CON_TRACE</code> tracing debugging output.</p> <p>This option is configured statically.</p>
<i>ECHO_LOG</i>	<p>Sets the <code>-e</code> command line option. Enter:</p> <ul style="list-style-type: none"> • 1 – to have all messages that are sent to the log display simultaneously to <code>stderr</code> while the current session of OpenSwitch is running. This is the default. • 0 – to have all messages sent only to the log and not display to <code>stderr</code>. <p>This option is configured statically.</p>
<i>FREEZE_CFG_ON_FAIL</i>	<p>Whether OpenSwitch locks all server and pool configurations (forbids all changes) when a network break is suspected between the companions during <code>CMP_FAIL_ACTION</code>. Enter:</p> <ul style="list-style-type: none"> • 0 – to have the OpenSwitch server continue servicing clients as if it were the only OpenSwitch running in a mutually-aware cluster. All configuration changes, including server and pool status changes, are permitted. This is the default. • 1 – to have the OpenSwitch server continue servicing clients, but forbids any changes to the server or pool configuration and status.

NAME	VALUE
<i>FULL_PASSTHRU</i>	<p>Sets the <i>-f</i> command line option.</p> <p>Full pass-through mode creates a pipeline for client requests to the server, and for results from the server to the client, which can reduce the overhead introduced by OpenSwitch in both directions, and can improve performance significantly. See “Performance” on page 15 for more information. Enter:</p> <ul style="list-style-type: none"> • 1 – to turn on full pass-through mode for communication between clients and the remote database servers. • 0 – to turn off full pass-through mode. All communications between the clients and the database servers are tracked by OpenSwitch so they can be restored properly during a failover. This is the default. <p>This option is configured statically.</p>
<i>HAFAILOVER</i>	<p>There is no command line equivalent for this option.</p> <p>This property is required to enable <i>HAFAILOVER</i>. Enter:</p> <ul style="list-style-type: none"> • 0 – to disable OpenSwitch from recognizing Adaptive Server HA events and error messages. This is the default. • 1 – to enable OpenSwitch to recognize Adaptive Server HA events and error messages. <p>This option is configured dynamically.</p>
<i>INTERFACES</i>	<p>Sets the <i>-I</i> (uppercase “i”) command line option.</p> <p>Enter an alternate location for the <i>sql.ini</i> (Windows) or <i>interfaces</i> (UNIX) file, rather than the default of <i>%SYBASE%\ini\sql.ini</i> (Windows) and <i>\$SYBASE/interfaces</i> (UNIX).</p> <p>This option is configured statically.</p>
<i>KEEPALIVE</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 1 – to turn <i>KEEPALIVE</i> on. • 0 – to turn <i>KEEPALIVE</i> off. This is the default. <p>See the Open Client documentation for CS_CON_KEEPALIVE for more information.</p> <p>This option is configured dynamically.</p>
<i>LOG_FILE</i>	<p>Sets the <i>-l</i> command line option.</p> <p>Enter the file name in which to save all messages to <i>log_file</i> rather than the default of <i>OpenSwitch.log</i>.</p> <p>This option is configured statically.</p>
<i>LOGIN_TIMEOUT</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the login timeout (CS_LOGIN_TIMEOUT property). The default is 60 seconds.</p> <p>This option is configured statically.</p>

NAME	VALUE
<i>MANUAL_SCRIPT</i>	<p>The path to the user-created manual script.</p> <hr/> <p>Note When you set <code>MUTUAL_AWARE=1</code>, the scripts for both OpenSwitch companions must perform the same action. This is because during a server failover, if you have set <code>SVR_FAIL_ACTION</code> to <code>MANUAL</code> or <code>CUSTOM</code>, only one of the companions executes the script that notifies the administrator or restarts the server.</p> <hr/> <p>See “Invoking custom and manual scripts” on page 111 for details about which exit codes to use in manual scripts.</p>
<i>MAX_LOG_MSG_SIZE</i>	<p>There is no command line equivalent for this option.</p> <p>Set this property when messages are bigger than 2048 characters. By default, OpenSwitch can log messages up to 2048 characters.</p> <p>This option is configured dynamically.</p>
<i>MAX_LOGSIZE</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the maximum size of the <i>log_file</i> (the default is 4194304). If <i>log_file</i> exceeds the size you set, OpenSwitch moves the current contents of <i>log_file</i> to a file named <i>currentfilename_old</i> and truncates the current log to 0 bytes.</p> <p>This option is configured statically.</p>
<i>MAX_PACKETSIZE</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the maximum size of a TDS packet. The default is 2048. Used to tune throughput across the network.</p> <p>This option is configured statically.</p> <hr/> <p>Warning! If the Adaptive Server max network packet size configuration parameter is set to 512 (the default), clients connections to OpenSwitch fail and the client receives this error message:</p> <p>The packet size (2048) specified at login time is illegal. Legal values are between 512 and 512.</p> <p>See “Connection refused” on page 30 for corrective actions.</p>
<i>MSGQ_SIZE</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the deferred event queue size (<code>SRV_S_DEFQUEUESIZE</code>) for the context of OpenSwitch. The default is 2048.</p> <p>See the <i>Open Server Server-Library/C Reference Manual</i> for more information.</p> <p>This option is configured statically.</p>
<i>MUTUAL_AWARE</i>	<p>Specifies whether to use mutually-aware OpenSwitch servers. Enter:</p> <ul style="list-style-type: none"> • 1 for a mutually-aware OpenSwitch. • 0 for a non-mutually-aware OpenSwitch, which is the default.

NAME	VALUE
<p><i>MUTUAL_CLUSTER</i></p> <p>Note Required only when <i>MUTUAL_AWARE</i>=1.</p>	<p>A string that represents a mutually-aware cluster. This string must be exactly the same on both OpenSwitch servers, and is used to name the configuration table in the <i>CFG_STORAGE</i> servers (see the [SERVER] section).</p> <p>When this parameter is not set, it defaults to “CLUSTER1”.</p>
<p><i>NET_FAIL_ACTION</i></p> <p>Note Required only when <i>MUTUAL_AWARE</i>=1. However, this parameter is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with <i>CMON</i> set to 1.</p>	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts.</p> <p>See “NET_FAIL_ACTION” on page 121 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>
<p><i>NOWAIT_ON_LOCKED</i></p>	<p>There is no command line equivalent for this option.</p> <p>If a client tries to log in or fail over to a server or pool that has a LOCKED state, the client connection is refused (by default) until either the status of the locked server or pool is changed to UP or DOWN, or the client times out and disconnects. If the status is set to UP, the client tries to connect to the server. If the status is set to DOWN, the client proceeds to the next available server or pool. To have the clients return immediately without being refused, set this option to 1; OpenSwitch returns an informational message to the clients describing the reason for the failure, without establishing the outgoing connections to the Adaptive Servers.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to not block clients trying to connect to a LOCKED server or pool. Instead, return error message 20103 to them immediately. If the client is an administrator (ADMIN_USER) trying to execute <i>rp_switch</i> to switch connections to a LOCKED server, error message 20104 is sent to the administrator instead. • 0 – to block clients trying to connect to a LOCKED server or pool. The blocked clients appears to have stopped responding until the affected server or pool is marked as either UP or DOWN. This is the default. <p>This option is configured dynamically.</p>

NAME	VALUE
<i>OPTIMIZE_TEXT</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> 1 – to optimize the processing of text or image data returned from the server by sending the data in “chunks” to the client. This option is used only in the processing of result rows where the result consists of a single column of the text or image datatype. This is the default. 0 – to conserve memory, which is preferable, especially when the text/image data received are usually small in size (less than 100 rows). <p>This option is configured dynamically.</p>
<i>PING_BINARY</i>	The absolute path to the system ping command. When this parameter is not set, it defaults to ping, which relies on the PATH environment variable to locate the correct binary.
<i>PING_RETRIES</i>	<p>The number of times that OpenSwitch should ping a server to rule out possible network problems. This parameter is used in numerous places, when the status must be known of a companion or Adaptive Server.</p> <p>The default is “1”.</p>
<i>PING_THREAD</i>	<p>Valid values are 0 (zero) or 1. The default is zero (0).</p> <p>When set to 1, <i>PING_THREAD</i> detects the failure when an Adaptive Server host or network stops running. Internally, the primary companion OpenSwitch checks to see if the entire network has failed or only the Adaptive Server host network. If the Adaptive Server host network has failed, the primary companion OpenSwitch is notified immediately and the action set for <i>SVR_FAIL_ACTION</i> is invoked.</p> <p>When set to zero (0), the primary companion OpenSwitch is notified of a network failure with Adaptive Server after only 8 minutes, which is the default TCP/IP <i>tcp_ip_abort_interval</i> configuration parameter.</p> <hr/> <p>Note Do not use this parameter to monitor the network between clients and OpenSwitch. If the network connection from the client to the OpenSwitch fails, the client detects the failure only when the <i>tcp_ip_abort_interval</i> time has elapsed. This is a kernel parameter that defaults to 8 minutes, but can be tuned to a lower value if the default is unacceptable.</p> <hr/>
<i>PING_WAIT</i>	<p>The number of seconds that the ping command should wait before returning a failure. This parameter is used in conjunction with <i>PING_RETRIES</i>, on platforms where the ping command blocks instead of returning right away.</p> <p>The default is 10 (seconds).</p>
<i>PRIMARY_COMPANION</i>	<p>Enter:</p> <ul style="list-style-type: none"> 0 – when this is not the primary companion. Zero (0) is the default. 1 – when this is the designated primary companion. A primary companion is responsible for writing to the Adaptive Server cluster tables.
Note Required only when <i>MUTUAL_AWARE=1</i> .	

NAME	VALUE
<i>RCM_AUTOSTART</i>	<p>Instruct OpenSwitch whether to start the replication coordination module (RCM). Enter:</p> <ul style="list-style-type: none"> • 0 – to not automatically start the RCM when OpenSwitch starts. This is the default value. • 1 – to start the RCM automatically when you start OpenSwitch. <p>This parameter is configured dynamically.</p>
<i>RCM_CFG_FILE</i>	<p>Enter the path where the RCM configuration file is located. This parameter has a null value if you do not specify a path.</p> <p>This parameter is configured statically.</p>
<i>RCM_LOG_FILE</i>	<p>Enter the path where the RCM log file should be created. This parameter has a NULL value if you do not specify a path.</p> <p>This parameter is configured statically.</p>
<i>RCM_PATH</i>	<p>Enter the path where OpenSwitch should look for the RCM executable.</p> <p>If you do not enter this path, and are using an RCM, OpenSwitch runs the RCM located in <i>\$OPENSWITCH/bin</i> on UNIX systems or in <i>%OPENSWITCH%\bin</i> on Windows systems; where <i>OPENSWITCH</i> is the installation directory.</p> <p>This parameter has a NULL value if you do not specify a path. This parameter is configured statically.</p>
<i>RCM_RETRIES</i>	<p>Enter how many times OpenSwitch should retry starting the RCM.</p> <p>If the RCM fails for reasons other than the user requesting that the RCM be shut down, OpenSwitch attempts to restart the RCM. If an unrequested shutdown of the RCM occurs within one minute of starting, OpenSwitch logs an error and does not attempt to restart the RCM.</p> <ul style="list-style-type: none"> • 0 – OpenSwitch does not attempt to restart the RCM. • Any numeric value – enter the number of times OpenSwitch should attempt to start the RCM. <p>This parameter is configured statically.</p>
<i>RCM_SECONDARY</i>	<p>Indicate to OpenSwitch whether the RCM it is launching is a primary or a secondary RCM. Valid values are zero (0, primary) or 1 (secondary). If you do not set this value, it remains at zero (0).</p> <p>This parameter is configured dynamically.</p>
<i>RESPONSE_TIMEOUT</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the timeout period for a response to a command (CS_TIMEOUT property). The default is 60 seconds.</p> <p>This option is configured statically.</p>

NAME	VALUE
<i>RMON</i>	<p>Note You must set this option for attributes under the [LIMIT_RESOURCE] section to take effect.</p> <p>Sets the -r command line option. Enter:</p> <ul style="list-style-type: none"> • 1 –to enable the OpenSwitch resource monitoring thread. • 0 – to disable the resource monitoring thread so resource governing is turned off. This is the default. See “Resource governing” on page 11 for more information. <p>This option is configured statically.</p>
<i>RMON_INTERVAL</i>	<p>There is no command line equivalent for this option.</p> <p>Set the frequency, in seconds, in which the resource monitoring thread awakens to check for offending connections. The default is 60 seconds.</p> <p>This option is configured statically.</p>
<i>RPC_SETFMT</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 0 – to have OpenSwitch return the results for an RPC in the order that is specified by the client return format. This is the default. • 1 – to have OpenSwitch return the results in the order that they are returned from the Adaptive Server. <p>This option is configured dynamically.</p>
<i>SEC_PRINCIPAL</i>	<p>Sets the -U command line option.</p> <p>Enter the principal name by which OpenSwitch is known to the security service provider. This sets the SRV_S_SEC_PRINCIPAL property in the Open Server if you are using a third-party security mechanism to check credentials. The default is NULL.</p> <p>This option is configured statically.</p>
<i>SERVER_NAME</i>	<p>Sets the -n command line option.</p> <p>Enter the name in the <i>interfaces</i> file on UNIX and the <i>sql.ini</i> file on Windows by which the OpenSwitch is to be referred.</p> <p>This option is configured statically.</p>
<i>SHOW_CONNECT_ERROR</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 1 – to have all ct_connect failure messages sent to the OpenSwitch log file. • 0 – the default. To not send ct_connect failure messages. <p>This option is configured dynamically.</p>

NAME	VALUE
<i>SHOW_SPID</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 0 – to prevent <i>rp_who</i> from showing the corresponding <i>spid</i> of a client connection to Adaptive Server. • 1 – to allow <i>rp_who</i> to show the corresponding <i>spid</i> of a client connection to Adaptive Server. <p>See <i>rp_who</i> on page 201 for more information on this command.</p> <p>A value of 1 introduces a small overhead to each new connection and should be set only when necessary. The default is 0.</p> <p>This option is configured statically.</p>
<i>SITE_PASSTHRU</i>	<p>There is no command line equivalent for this option.</p> <p>Determines whether connections from site handlers can be passed through OpenSwitch. Enter:</p> <ul style="list-style-type: none"> • 1 – to have connections pass through as normal client connection. • 0 – to have connections treated as from an administrator and not establish the outgoing connection. <p>This option is configured dynamically.</p>
<i>SQL_WRAP</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the maximum number of characters per line for a SQL statement. Used when OpenSwitch writes SQL statements to a log file if the language debug logging option is turned on.</p> <p>This option is configured dynamically.</p>
<i>SRV_TRACE</i>	<p>Sets the -s command line option.</p> <p>Enter server-level tracing flags. The output is placed into the location of LOG_FILE. See “Using command line options” on page 49 for available settings.</p> <p>This option is configured statically.</p>
<i>STACKSIZE</i>	<p>Sets the -S command line option.</p> <p>Enter the stack size for each thread. The default is 40960.</p> <hr/> <p>Note On 64-bit platforms, double this value to prevent a stack overflow.</p> <hr/> <p>This option is configured statically.</p>

NAME	VALUE
<i>SUPPRESS_CHARSET</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 1 – to suppress the 5704 error message from the server and the informational message reported by OpenSwitch when the OpenSwitch character set is different from the client application character set. The default is 1. • 0 – to log a warning message in the LOG_FILE (and also sent to <i>stderr</i> if ECHO_LOG=1) when a client connects to OpenSwitch with a character set different from the one defined in CHARSET. <p>This option is configured dynamically.</p>
<i>SUPPRESS_DBCTX</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 0 – to allow the database context change error message (5701). • 1 – suppresses the database context change error message (5701). The default is 1. <p>This option is configured dynamically.</p>
<i>SUPPRESS_LANG</i>	<p>There is no command line equivalent for this option.</p> <ul style="list-style-type: none"> • 0 – to allow the database language change error message (5703). • 1 – to suppress the language change error message (5703) from the server. The default is 1. <p>This option is configured dynamically.</p>
<i>SVR_FAIL_ACTION</i> Note Required only when MUTUAL_AWARE=1.	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when an Adaptive Server fails to respond in a timely manner, or when the Adaptive Server host cannot be pinged by either OpenSwitch server in a cluster.</p> <p>See “SVR_FAIL_ACTION” on page 122 for more information. See “User-specified actions” on page 113 for details about the available actions.</p>

NAME	VALUE
<i>SWITCH_AT_LOGIN_TIMEOUT</i> <i>T</i>	<p>There is no command line equivalent for this option.</p> <p>This option works in conjunction with LOGIN_TIMEOUT and RESPONSE_TIMEOUT. If a new connection takes longer than the LOGIN_TIMEOUT value to complete, or an existing connection takes longer than RESPONSE_TIMEOUT to receive a reply back from the Adaptive Server, this option tells OpenSwitch whether to switch the affected client to the next available Adaptive Server.</p> <hr/> <p>Note If this option is set to 1, you must ensure that LOGIN_TIMEOUT and RESPONSE_TIMEOUT are both set to realistic values that, when exceeded, indicate a real problem in the server that warrants a failover.</p> <hr/> <p>Enter:</p> <ul style="list-style-type: none"> • 0 – to have the connection try the next server only if a connection failure error is received (CS_SV_COMM_FAIL). The default is zero (0). • 1 – to have the connection try the next server if there is a time-out error. OpenSwitch fails over to a backup server, even if the network is just slow or Adaptive Server is too busy to respond. <p>This option is configured dynamically.</p>
<i>TCP_NODELAY</i>	<p>There is no command line equivalent for this option. Enter:</p> <ul style="list-style-type: none"> • 1 – to turn TCP_NODELAY on. • 0 – to turn TCP_NODELAY off. This is the default. <p>See the Open Client documentation for CS_CON_TCP_NODELAY for more information.</p> <p>This option is configured dynamically.</p>
<i>TEXTSIZE</i>	<p>There is no command line equivalent for this option.</p> <p>Enter the maximum size, in bytes, of text or image columns that can be handled by OpenSwitch. Due to the way in which Open Server handles large text columns, memory must be allocated in which to hold the entire column while a result set is processed. Although this memory is held only long enough for the entire result set to be returned to the client, setting this option to an unusually large value can affect the total memory consumed by OpenSwitch.</p> <p>This option is configured dynamically.</p>
<i>TRUNCATE_LOG</i>	<p>Sets the -T command line option. Enter:</p> <ul style="list-style-type: none"> • 1 – to truncate the output logging file prior to start-up. For more information on the output logging file, see LOG_FILE in this table. • 0 – to append the log file rather than truncate it. This is the default. <p>This option is configured statically.</p>

NAME	VALUE
<i>UPDATE_CFG</i>	When set to 1, the configuration file is updated each time a reconfiguration takes place. Zero (0) is the default setting. You must run mutual-aware support with <code>UPDATE_CFG=1</code> .
<i>USE_AND_TO_POOL_ATTRIB</i>	<p>There is no command line equivalent for this option.</p> <p>Indicates whether all or only one of the connection attributes in the [POOL] section are to be satisfied by a client.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to specify that only a client that satisfies all the connection attributes under a pool is allowed to use the pool. • 0 – to specify that only a client that satisfies any one of the connection attributes under a pool is allowed to use the pool. This is the default. <p>For example:</p> <pre>[CONFIG] USE_AND_TO_POOL_ATTRIB = 1 [POOL=POOL1:MODE=CHAINED,CACHE=0] servers: ASE1 connections: username: john appname: isql</pre> <p>In this example, if <code>USE_AND_TO_POOL_ATTRIB</code> is set to 1, only clients with the user name of “john” and the application name of “isql” are allowed to use <code>POOL1</code>.</p> <p>If <code>USE_AND_TO_POOL_ATTRIB</code> is set to 0, any client with either the user name of “john” or the application name “isql” is allowed to use <code>POOL1</code>.</p> <p>This option is configured dynamically.</p>

NAME	VALUE
<i>USE_AND_TO_RMON_ATTRIB</i>	<p>There is no command line equivalent for this option.</p> <p>Indicates whether the connection attributes in the [LIMIT_RESOURCE] section can use “AND” or “OR.”</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1 – to specify that only a client that satisfies all the attributes under the [LIMIT_RESOURCE] sections is subject to the resource governing rules. • 0 – to specify that any client that satisfies any one of the attributes under the [LIMIT_RESOURCE] section is subject to resource governing rules. This is the default. <p>For example:</p> <pre>[CONFIG] RMON= 1 USE_AND_TO_RMON_ATTRIB=1 [LIMIT_RESOURCE:ACTION=KILL,BUSY=60] username: john appname: isql hostname: machine type:client</pre> <p>In the example, set to 1, only connections from the user “john” running the application “isql” from the host “machine” and not a site-handler are monitored and disconnected by the resource manager if their transactions take longer than 60 seconds to complete.</p> <p>If the example used “0”, all connections from the user “john” or from the application “isql” or from the host “machine” or of the client type are monitored and disconnected by the resource manager if their transactions take longer than 60 seconds to complete.</p> <p>This option is configured dynamically.</p>
<i>USE_DONEINPROCS</i>	<p>When set to 1, OpenSwitch sends the TDS DONEPROC and DONEINPROCS tokens received from the Adaptive Server to connected clients. The default is zero (0).</p>
<i>USERNAME_PASSWORD_ENCRYPTED</i>	<p>There is no command line equivalent for this option.</p> <p>Enter:</p> <ul style="list-style-type: none"> • 1– to indicate user name and password encryption. OpenSwitch expects <i>ADMIN_USER</i>, <i>ADMIN_PASSWORD</i>, <i>COORD_USER</i>, <i>COORD_PASSWORD</i>, <i>CMON_USER</i>, and <i>CMON_PASSWORD</i> to be encrypted. • 0 – to specify no user name and password encryption. OpenSwitch expects clear text for <i>ADMIN_USER</i>, <i>ADMIN_PASSWORD</i>, <i>COORD_USER</i>, <i>COORD_PASSWORD</i>, <i>CMON_USER</i>, and <i>CMON_PASSWORD</i>. <p>This option is configured statically.</p>

Example

In this example, “LOG_FILE” identifies that the -l flag is being set, and *OpenSwitch.log* is the value to use.

```
[CONFIG]
LOG_FILE = OpenSwitch.log
...
COORD_MODE      = ALWAYS
RCM_AUTOSTART   = 1
RCM_RETRIES     = 3
RCM_PATH        = /OpenSwitch-15_0/bin/rcm
RCM_CFG_FILE    = /OpenSwitch-15_0/config/rcm1.cfg
RCM_LOG_FILE    = /OpenSwitch-15_0/logs/rcm.err
RCM_SECONDARY   = 0
...
...
...
```

[SERVER]**Description**

This section of the configuration file defines the set of remote servers available for use within pools. Use this section to predefine the disposition of a server before you start OpenSwitch.

Complete the [SERVER] parameters once for each remote server to be accessed.

Format

```
[SERVER=SERVER_NAME]
OPTION = VALUE
OPTION = VALUE
CMON_USER = CMON user name
CMON_PASSWORD = CMON password
```

Parameters

- *SERVER_NAME* – the name of the remote server as listed in the *\$SYBASE/interfaces* file on UNIX and in the *%SYBASE%\sql.ini* file on Windows.
- A value of *DEFAULT* sets the default values for any server not explicitly listed in a this section.
- *OPTION* and *VALUE* – additional parameters specific to *SERVER_NAME*. Table 4-2 on page 84 shows valid input for these parameters, which are entered in the format:

```
OPTION=VALUE
```

Table 4-2: [SERVER] OPTION and VALUE settings

OPTION	VALUE
CFG_STORAGE	<p>When this property is set for an Adaptive Server, that Adaptive Server is used to store the configuration information of the mutually-aware OpenSwitch clusters.</p> <ul style="list-style-type: none"> 0 – do not use this Adaptive Server to store the configuration information of the mutually-aware OpenSwitch clusters. This is the default value. 1 – store the configuration information of the mutually-aware OpenSwitch clusters on this Adaptive Server. <p>In a mutually-aware setup, you must include this parameter under the two Adaptive Server entries; for example:</p> <pre>[SERVER=ASE1] STATUS=UP CFG_STORAGE=1 [SERVER=ASE2] STATUS=UP CFG_STORAGE=1</pre> <p>See Chapter 5, “Using Mutually-aware OpenSwitch Servers,” for more information.</p>
STATUS	<p>Enter the status of the server, which is the state of the server within OpenSwitch:</p> <ul style="list-style-type: none"> UP – the server is currently available. DOWN – the server is unavailable, and will not be considered for use by any new client connections established to the OpenSwitch. LOCKED – the server is available, but any new incoming connections or existing connections switching to the server are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to client applications to have stopped responding until the server is unlocked. <p>Servers can also have a PRE_UP, PRE_DOWN, or PRE_LOCKED status. These are states specific to mutually-aware OpenSwitch servers. Never manually set a server’s status to these states. For more information, see “Server state” on page 20.</p> <hr/> <p>Note You cannot start OpenSwitch with a server in the LOCKED state. To start OpenSwitch, all servers must be either UP or DOWN. Once OpenSwitch starts, use <code>rp_server_status</code> to change the server STATUS to LOCKED.</p>

OPTION	VALUE
TYPE	<p>If the server in a pool is HA-enabled, enter HA for this parameter. If the server is not HA-enabled, you can exclude this parameter or leave it blank.</p> <p>If the server is in an Adaptive Server HA cluster, verify this property is set to “HA” in that Adaptive Server [SERVER] sections. For example:</p> <pre> [CONFIG] HAFAILOVER = 1 CMON_USER = cmon_user CMON_PASSWORD = cmon_password [SERVER = nlatke_ASE1] STATUS = UP TYPE = HA </pre> <ul style="list-style-type: none"> CMON_USER – the user login used by the connection monitor (CMON) thread to connect to the back-end server. This must be an existing, valid login on each of the Adaptive Servers being used. Verify this user has basic privileges and can issue a “wait for delay” query to the remote data serve. <p>This value supersedes the value you set for <i>CMON_USER</i> in the [CONFIG] section. If you do not set a value here, OpenSwitch uses the value you set for <i>CMON_USER</i> in the [CONFIG] section.</p> <p>Also, if the <i>CMON_USER</i> is not a valid Adaptive Server user, the client can be left in an undefined state when OpenSwitch is configured for failover mode and a failover occurs to the secondary Adaptive Server. Be sure to specify a valid CMON user name and that the user is a valid Adaptive Server user.</p> <p>If the <i>USERNAME_PASSWORD_ENCRYPTED</i> option is set to 1, this should contain the encrypted string of the CMON user name. To encrypt a user name, use the -E or -p command line options.</p> CMON_PASSWORD – the password for the CMON_USER. The default is an empty string. <p>This value supersedes the value you set for <i>CMON_PASSWORD</i> in the [CONFIG] section. If you do not set a value here, OpenSwitch uses the value you set for <i>CMON_PASSWORD</i> in the [CONFIG] section.</p>
Examples	<p>In the example above, the latest OpenSwitch configuration is always stored in the syso_<cluster> table in ASE1 and ASE2 (as long as they are running and connected). Therefore, when a mutually-aware OpenSwitch server fails to connect to its companion OpenSwitch for the latest configuration information, it queries both ASE1 and ASE2 to retrieve that information.</p> <pre> [SERVER=ASE1] </pre>

```
STATUS = UP
CFG_STORAGE = 1
TYPE = HA
CMON_USER = ASE1usr
CMON_PASSWORD = ASE1pwd

[SERVER=ASE2]
STATUS = UP
CFG_STORAGE = 1
CMON_USER = sa
CMON_PASSWORD = sa
```

[POOL]

Description Complete this section once for each Adaptive Server pool you want defined within OpenSwitch. The section includes the pool name, any attributes of the pool, the set of servers contained within the pool, and the set of connections that the pool serves.

Format [POOL=POOL_NAME[: OPTION=VALUE[, OPTION=VALUE]]]
[servers:]
SERVER_NAME
[SERVER_NAME ...]
[connections:
attribute: regex [, regex]
[attribute: regex [, regex] ...]]

- Parameters
- POOL_NAME – a unique pool name up to 30 characters in length.
 - OPTION and VALUE – a configuration option specific to POOL. Table 4-3 shows valid input for these parameters, which are entered in the format OPTION = VALUE.

Table 4-3: [POOL] OPTION and VALUE settings

OPTION	VALUE
MODE	<div>The operational mode of the pool:<ul style="list-style-type: none">• CHAINED – all clients qualified to use the pool are channeled to the first server in the pool that has a status of UP or LOCKED. When the first server fails or has a status of DOWN, the clients are channeled to the next server within the same pool with a status of UP or LOCKED.• BALANCED – all qualifying clients are distributed among all the servers within the pool in a round-robin fashion. When one of the servers fails or is marked as DOWN, its clients are then distributed among the other servers within the same pool.</div>

OPTION	VALUE
STATUS	<p>The status of the pool:</p> <ul style="list-style-type: none"> UP – the pool is up and ready for connections. All new and failover clients that satisfy its connection attributes are channeled to it. DOWN – the pool is currently unavailable. All new and failover clients skip over it to proceed to the next UP or LOCKED pool. LOCKED – the pool is designated to serve the connections, but is temporarily unavailable. By default, the qualifying clients are blocked until the pool is marked as UP or DOWN. However, if <i>NOWAIT_ON_LOCKED</i> is set to 1, the clients immediately receive a descriptive error message from OpenSwitch. <p>Pools can also have a PRE_UP, PRE_DOWN, PRE_LOCKED, or SUSPENDED status. These states are specific to pools in a mutually-aware implementation. Never manually set a pool's status to these states. For more information, see "Pool states" on page 26.</p>
CACHE	<p>The number of seconds that connections are cached following a user disconnect. See "Using connection caching" on page 34.</p> <p>The value is an integer in seconds.</p> <ul style="list-style-type: none"> [servers: <div style="margin-left: 20px;"> <code>server_name=status</code> <code>server_name=status</code> </div> <p>"servers" is an optional [POOL] subsection that lists the servers in the pool, and optionally the server's status.</p> <i>server_name</i> – the remote server defined in the <i>\$SYBASE/interfaces</i> file on UNIX and <i>%SYBASE%\ini\sql.ini</i> on Windows. <hr/> <p>Note During OpenSwitch start-up, no validation is done to verify whether or not these servers exists.</p> <hr/> <ul style="list-style-type: none"> <i>status</i> – (optional) enter the status of the specified server (<i>server_name</i>) at the pool level. When you specify the server status in the [SERVER] section of the configuration file, the status is applied globally. When you supply a <i>status</i> here, the value overwrites the server's global status with the status you enter for this server in this pool. This allows the same server to have a different status in different pools, allowing you to failover selected applications to back-up servers. Enter: <ul style="list-style-type: none"> UP – the server is currently available. DOWN – the server is unavailable, and will not be considered for use by any new client connections established to the OpenSwitch.

- **LOCKED** – the server is available, but any new incoming connections or existing connections switching to the server are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to client applications to have stopped responding until the server is unlocked.

Servers can also have a PRE_UP, PRE_DOWN, or PRE_LOCKED status; however, you should never manually set a server's status to these states. For more information, see “Server state” on page 20.

Note You cannot start OpenSwitch with a server in the LOCKED state. To start OpenSwitch, all servers must be either UP or DOWN. Once OpenSwitch starts, use `rp_server_status` to change the server STATUS to LOCKED.

For more information, see “Server state” on page 20.

- [connections:
attribute: regex [, *regex*]
[attribute: regex [, *regex*] ...]]

Optional [POOL] subsection that lists the attributes that define a set of connections that use this pool.

- *attribute* – the name of a connection attribute. Valid attributes are:

Attribute	Description
username	Match the user name supplied by the Open Client connection.
appname	Match the application name declared by the Open Client connection.
hostname	Match the host name on which the Open Client connection is running.
type	Match the type of incoming connection as either a site connection or a client connection. Possible values are: <ul style="list-style-type: none"> • Client – for regular client connections. • Site – for site-handler connections.

- *regex* – the value for the specified attribute. This can also be a standard SQL-style extended regular expression that describes values for a given attribute that uses this pool. You can use wildcard expressions to specify a range or group of acceptable attribute values.

If you omit the [connections] subsection of the pool configuration, all connections are a match for the pool. However, the order in which pools are defined is important; a connection is routed to the first matching pool. Therefore, Sybase suggests that you keep a “catch-all” pool without a [connections] section at the end of the pool list.

Examples

In this example, any client that logs in to OpenSwitch with the user name of “john” or the application name “isql” (depending on the *USE_AND_TO_POOL_ATTRIB* setting) is first channeled to the Adaptive Server ASESrv1, which has a status of UP in POOL1.

If the client fails to connect to ASESrv1, or it has been marked as DOWN explicitly by the coordination module or the administrator, the clients are channeled to ASESrv3 because ASESrv2 is marked as DOWN in POOL1.

When a client disconnects, its connection to the Adaptive Server is cached for as long as five minutes. If the same user with the same password reconnects within that period, he or she can reuse the cached connection, thereby saving the overhead of creating a new outgoing connection.

In this example, “john” is the *regex* value for the “username” *attribute*, and “isql” is the *regex* value for the “application” *attribute*:

```
[POOL=POOL1:MODE=CHAINED,CACHE=300]
servers:
    ASESrv1=UP
    ASESrv2=DOWN
    ASESrv3=UP
connections:
    username: john
    appname: isql
```

[LIMIT_RESOURCE]

Description

The options in this section indicate the resource to be constrained and the maximum amount of the resource that can be consumed by a given connection.

Note You must set the RMON parameter in the [CONFIG] section to have the options in the [LIMIT_RESOURCE] section take effect.

Format

```
[LIMIT_RESOURCE:ACTION=[{KILL|CANCEL}][BUSY=value]
  attribute: regex [, regex]
  [attribute: regex [, regex]]
```

Parameters

- **ACTION {KILL | CANCEL}**– select the action to be taken by the resource monitor upon detecting a connection that has exceeded one or more resource limits. Enter:
 - **KILL** – to disconnect the connection from the server with no warning.
 - **CANCEL** – to cancel the connection and send a message to the client.
- **value** – enter a value, in seconds, to limit connections to a maximum amount of busy-time. Busy-time includes the time it takes to process a query and return the results to the caller. Connections currently in the middle of a transaction but not actively processing a query are still considered busy.
- **attribute** – the name of a connection attribute. Valid attributes are:

Attribute	Description
username	Match the user name supplied by the Open Client connection.
appname	Match the application name declared by the Open Client connection.
hostname	Match the host name on which the Open Client connection is running.
type	Match the type of incoming connection as either a site connection or a client connection. Possible values are: <ul style="list-style-type: none"> • Client – for regular client connections. • Site – for site-handler connections.

- **regex** – the value for the specified attribute. This can also be a standard SQL-style extended regular expression that describes values for a given attribute that uses this pool. You can use wildcard expressions to specify a range or group of acceptable attribute values.

Examples

This example indicates that the *attribute: regex* pairs that follow the header are to be constrained to a single query lasting no longer than 5 minutes (300 seconds). You can specify multiple attributes in the configuration file:

```
[LIMIT_RESOURCE:ACTION=KILL,BUSY=60]
  username: john
  appname: isql
  hostname: machine
  type: client
```

[COMPANION]

Description	Name of the OpenSwitch companion, and the administrator user name and password used to make a connection. For example:
Format	<pre>[COMPANION=companion_name] admin_user=ADMIN_USER admin_password=ADMIN_PASSWORD</pre>
Parameters	<ul style="list-style-type: none">• <i>companion_name</i> – the <i>SERVER_NAME</i> from the companion OpenSwitch configuration file. This is a static parameter; if you change this value, you must restart the OpenSwitch server for the change to take effect.• <i>admin_user</i> – the <i>ADMIN_USER</i> from the companion OpenSwitch configuration file. This is a dynamic parameter, which means that you can change the value using <i>rp_cfg</i> while OpenSwitch is running, and the changes take effect immediately.• <i>admin_password</i> – the <i>ADMIN_PASSWORD</i> from the companion OpenSwitch configuration file. This is a dynamic parameter, which means that you can change the value using <i>rp_cfg</i> while OpenSwitch is running, and the changes take effect immediately
Examples	<pre>[COMPANION=OSW2] admin_user=sa admin_password=sa</pre>

Using Mutually-aware OpenSwitch Servers

This chapter describes the support in OpenSwitch 15.0 and later for mutually-aware OpenSwitch servers, and how to implement this functionality.

Topic	Page
Introduction	93
Requirements	94
Configuring OpenSwitch servers to be mutually aware	99
OpenSwitch mutually-aware operations	109
Invoking custom and manual scripts	111

Introduction

OpenSwitch supports a redundant environment with two “mutually-aware” OpenSwitch servers that serve the same pools of two Adaptive Servers. Each OpenSwitch server is aware of the other OpenSwitch server and whether the Adaptive Servers for which that server is responsible are ready to accept client connections.

For example, let’s say you have one OpenSwitch server (OSW1) connected to one Adaptive Server (Server1). You have another OpenSwitch server (OSW2) and another Adaptive Server (Server2) set up for failovers. OSW1 fails and its connections to Server1 fail over to OSW2. Then Server1 fails over to Server2. In versions earlier than 15.0, when you restarted OSW1, it would continually try to reconnect to Server1 because it had no knowledge that Server1 was not running.

In a mutually-aware OpenSwitch implementation, when OSW1 restarts, it queries the companion OpenSwitch server (OSW2), which tells it that Server1 has failed, therefore, it does not try to reconnect to the unavailable server.

To ensure that this setup works properly, both OpenSwitch servers must:

- Have the same knowledge about each pool's and server's status at any given time.
- Store and retrieve the redundancy information from a set of locations that are accessible to all components in the OpenSwitch implementation to ensure that the effective status persists between OpenSwitch restarts.

Before you install and configure mutually-aware OpenSwitch servers, be sure to read this chapter and make sure that you review the mutually-aware OpenSwitch server “Requirements” on page 94.

Requirements

This section lists the installation, configuration, and use requirements for a successful OpenSwitch mutually-aware implementation. The remainder of this chapter discusses all requirement in more detail.

Installation

A mutually-aware OpenSwitch implementation requires that you install:

- Two mutually aware, companion OpenSwitch servers, preferably on a different host than the Adaptive Servers. Both mutually aware OpenSwitch servers within the same cluster regard each other as companions and are both aware of each other's state and the state of the other servers.

Note You can also use two coordination modules or two replication coordination modules, but these OpenSwitch CM applications are optional. See the *OpenSwitch Coordination Module Reference Guide*.

- Two Adaptive Servers, which may be configured for high-availability.

Configuration and use

When you complete the OpenSwitch installation, you can configure OpenSwitch immediately using the GUI configuration tool, or you can configure the product manually by editing the OpenSwitch configuration file (see “Editing the OpenSwitch configuration file” on page 60). You can also run the standalone configuration tool after installation (see “Starting the Configuration tool,” in Chapter 3 of the *OpenSwitch Installation Guide*).

Regardless of the configuration method used, you must configure OpenSwitch for mutually-aware behavior:

- Configuration GUI – select Use Mutual Aware Support? in the OpenSwitch Configuration dialog box.
- Configuration file – enter 1 for the *MUTUAL_AWARE* parameter (*MUTUAL_AWARE=1*).

You must also complete specify the action OpenSwitch should take when certain failure types occur. See “Failure types” on page 112.

Requirements

When you configure and use mutually-aware OpenSwitch servers, keep these requirements in mind:

- Server and pool names are limited to a maximum of 31 characters.
- Both OpenSwitch servers must have the same pool and server configurations in their configuration files. When they are not the same, the OpenSwitch that starts first takes precedence.
- The following [CONFIG] parameters must be the same in both OpenSwitch server configuration files:
 - *MUTUAL_AWARE*
 - *MUTUAL_CLUSTER*
 - *UPDATE_CFG*
 - *SVR_FAIL_ACTION*
 - *NET_FAIL_ACTION*
 - *CMP_FAIL_ACTION*
 - *CMON_FAIL_ACTION*
 - *FREEZE_CFG_ON_FAIL*

- *USERNAME_PASSWORD_ENCRYPTED*

Each of these parameters is checked and compared when a mutually-aware OpenSwitch server starts, and monitored by the mutually-aware OpenSwitch server during runtime. If there is a difference in parameter values between the companions during startup, the OpenSwitch that is being started fails. If any parameter is found to be different between mutually-aware companions at runtime, both companions are suspended until the difference is resolved and the administrator issues `rp_go` on each companion. In either case, an error message is logged regarding the dissimilar parameters to help you diagnose and correct the problem.

- Set *UPDATE_CFG* to 1 when you set *MUTUAL_AWARE* to 1.
- When starting mutually-aware OpenSwitch servers, verify that the first server starts properly before starting the next server.
- When it is preferable for clients to fail over from one OpenSwitch server to the companion, you must set up the *sql.ini* file (Windows), *interfaces* file (UNIX), or LDAP directory service to facilitate this failover. For the *sql.ini* or *interfaces* file, enter the second OpenSwitch query address under the first OpenSwitch name as a secondary address. For example:

```
OSW1
query    tcp    ether    <host>    4405
query    tcp    ether    <host>    4406

OSW2
query    tcp    ether    <host>    4406
query    tcp    ether    <host>    4405
```

Note When you configure a mutually-aware OpenSwitch server using the GUI configuration tool, these entries are created for you automatically.

When you configure a mutually-aware OpenSwitch server by manually editing the configuration files, you must manually create these entries in the *sql.ini* (Windows) or *interfaces* (UNIX) file.

- When the data in the OpenSwitch configuration file is more recent than the configuration used by OpenSwitch when it last ran, start the OpenSwitch server with the “-O” parameter to override the data in the Adaptive Server configuration tables with the information from the OpenSwitch configuration file. For example:

On Windows:

```
%OPENSWITCH%\bin\OpenSwitch.bat -c \
%OPENSWITCH%\config\<config_file_name>.cfg -O
```

On UNIX:

```
$OPENSWITCH/bin/OpenSwitch -c \
$OPENSWITCH/config/<config_file_name>.cfg -O
```

- The configuration table is created in the default database of the CMON user in the Adaptive Servers where `CFG_STORAGE=1`. When the default database is not explicitly set, it defaults to “master.”
- When an Adaptive Server has not been used for a long time and may have legacy information from an outdated OpenSwitch setup, the administrator can truncate the table before starting the new OpenSwitch server, or start the OpenSwitch server with the “-O” parameter to override the information in the OpenSwitch configuration file.
- Only the first two pools defined in the OpenSwitch configuration file are mutually aware.

Note Sybase recommends that you not have other pools in a mutually-aware OpenSwitch, unless the pool is a “catch-all pool” that is not crucial to production.

Pools other than the first two pools in the configuration file assume their original behavior. However, if failover occurs in either of the first two pools, a mutually-aware OpenSwitch acts in the best interest of the first two pools. This may involve shutting down to allow the clients to fail over to the companion OpenSwitch server in case of a total network failure.

Verify that all the pools are defined identically in the primary and secondary companion OpenSwitch servers; that is, pools with the same name must contain the same Adaptive Servers and have the same connection attributes.

- Each of the first two pools must contain two servers. The two servers must be defined the same in both pools, but defined in reverse order:

```
[POOL=POOL1]
servers:
    ASE1
    ASE2 [POOL=POOL2]
servers:
    ASE2
    ASE1
```

- Currently, only two OpenSwitch servers are supported in a mutually-aware cluster; that is, there can be only one companion OpenSwitch for each mutually-aware OpenSwitch server. Both OpenSwitch servers in the cluster must be mutually aware.

If one of the OpenSwitch companions does not have *MUTUAL_AWARE* set to 1, the other OpenSwitch companion marks that OpenSwitch server as “down.” This inconsistency is noted in the error log. Until the non-mutually-aware OpenSwitch server is restarted with *MUTUAL_AWARE* set to 1, it receives no communication from the companion that does have *MUTUAL_AWARE* set to 1, which acts as the only mutually-aware OpenSwitch server running in the cluster.

Note Sybase recommends that you never have two OpenSwitch servers use the same pools of servers, or be the failover entries for each other in the *sql.ini* (Windows) or *interfaces* (UNIX) file without being mutually aware. Always set *MUTUAL_AWARE* to 1 in both companion OpenSwitch server configuration files.

- A mutually-aware configuration does not require a coordination module (CM) or replication coordination module (RCM) to run. However, a mutually-aware implementation can work with a CM or RCM solution.
- If you are using a CM or RCM, set *SVR_FAIL_ACTION* to DEFAULT. Specifically, you cannot run custom or manual scripts for server failure when you are using a CM or RCM because each CM and RCM is coded differently and its failover procedure may contradict the actions invoked by a custom or manual script.
- If you use a system router, it should not block system commands, such as “ping.” OpenSwitch uses ping to monitor the network between mutually-aware OpenSwitch companions and between OpenSwitch and the Adaptive Servers. A ping failure can be incorrectly interpreted as a network failure.
- Start the primary OpenSwitch first, then start the secondary (companion) OpenSwitch. See Chapter 3, “Starting and Stopping OpenSwitch and RCMs.”

Configuring OpenSwitch servers to be mutually aware

To implement mutually-aware OpenSwitch servers, install and configure two OpenSwitch servers.

Note In a production environment, the Adaptive Servers are generally installed on a different host than the OpenSwitch servers. When both OpenSwitch servers are installed on the same host machine, each OpenSwitch server must be installed in a different Sybase directory.

When you install OpenSwitch and configure it using the configuration tool, the process creates two files:

- *sql.ini* (Windows) or *interfaces* (UNIX) – contains information about the network locations of servers. The file contains at least one entry that specifies the network connection information for the default server. This file is saved in the *SYBASE* root directory.
- *<OpenSwitch_server_name>.cfg* – data gathered during OpenSwitch configuration. By default, the file is given the name of the OpenSwitch server being configured, followed by the *.cfg* file name extension.

Note You can also manually create and edit the OpenSwitch configuration file.

When you specify during configuration that an OpenSwitch server should be mutually aware, the configuration process also creates a mutually-aware configuration table in both Adaptive Servers, which can be read by both OpenSwitch servers.

Configuration file parameters

Table 5-1 lists the OpenSwitch configuration parameters specific to mutually-aware OpenSwitch servers. Parameters are listed alphabetically within each configuration file section for ease of reference.

Table 5-1: Mutually-aware configuration parameters

Section	Parameter	Description
[CONFIG]	<p><i>CMON_FAIL_ACTION</i></p> <hr/> <p>Note Required only when <code>MUTUAL_AWARE=1</code>. However, this parameter is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with <i>CMON</i> set to 1.</p> <hr/>	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>The selected action is invoked only when the <i>CMON</i> configuration parameter is set to 1 (the recommended setting), and when the <i>CMON</i> thread that monitors the health of the Adaptive Server fails to start. See Table 4-1 on page 65 for information about the <i>CMON</i> parameter.</p> <p>See “<i>CMON_FAIL_ACTION</i>” on page 119 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>
[CONFIG]	<p><i>CMP_FAIL_ACTION</i></p> <hr/> <p>Note Required only when <code>MUTUAL_AWARE=1</code>.</p> <hr/>	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when network connectivity is lost to the companion OpenSwitch in a mutually-aware setup. Once the network is restored and the connection to the companion is re-established, the two OpenSwitch servers synchronize their configurations.</p> <p>See “<i>CMP_FAIL_ACTION</i>” on page 120 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>
[CONFIG]	<p><i>CUSTOM_SCRIPT</i></p> <hr/>	<p>The path to the user-created script to invoke.</p> <hr/> <p>Note When <code>MUTUAL_AWARE=1</code>, the scripts for both OpenSwitch companions must perform the same action. When a server fails over and <i>SVR_FAIL_ACTION</i> is set to <code>MANUAL</code> or <code>CUSTOM</code>, only one of the companions executes the script that notifies the administrator or restarts the server.</p> <hr/> <p>See “User-specified actions” on page 113 for details about which exit codes to use in custom scripts.</p>

Section	Parameter	Description
[CONFIG]	<i>FREEZE_CFG_ON_FAIL</i>	<p>Whether OpenSwitch locks all server and pool configurations (forbids all changes) when a network break is suspected between the companions during <i>CMP_FAIL_ACTION</i>. Enter:</p> <ul style="list-style-type: none"> • 0 – allows the OpenSwitch server to continue servicing clients as if it were the only OpenSwitch running in a mutually-aware cluster. All configuration changes, including server and pool status changes, are permitted. This is the default. • 1 – allows the OpenSwitch server to continue servicing clients, but forbids any changes to the server or pool configuration and status.
[CONFIG]	<i>MANUAL_SCRIPT</i>	<p>The path to the user-created manual script.</p> <p>See “User-specified actions” on page 113 for details about which exit codes to use in manual scripts.</p>
[CONFIG]	<i>MUTUAL_AWARE</i>	<p>Specifies whether to use mutually-aware OpenSwitch servers. Enter:</p> <ul style="list-style-type: none"> • 1 for a mutually-aware OpenSwitch. • 0 for a non-mutually-aware OpenSwitch, which is the default.
[CONFIG]	<i>MUTUAL_CLUSTER</i> <hr/> Note Required only when <i>MUTUAL_AWARE</i> =1.	<p>A string that represents a mutually-aware cluster. This string must be exactly the same on both OpenSwitch servers, and is used to name the configuration table in the <i>CFG_STORAGE</i> servers (see “[SERVER]” on page 83).</p> <p>When this parameter is not set, it defaults to “CLUSTER1”.</p>
[CONFIG]	<i>NET_FAIL_ACTION</i> <hr/> Note Required only when <i>MUTUAL_AWARE</i> =1.	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts.</p> <p>See “NET_FAIL_ACTION” on page 121 for more information about this parameter. See “User-specified actions” on page 113 for details about each action.</p>
[CONFIG]	<i>PING_BINARY</i>	<p>The absolute path to the system ping command. When this parameter is not set, it defaults to ping, which relies on the PATH environment variable to locate the correct binary.</p>

Section	Parameter	Description
[CONFIG]	<i>PING_RETRIES</i>	<p>The number of times that OpenSwitch should ping a server to rule out possible network problems. This parameter is used in numerous places, when the status must be known of a companion or Adaptive Server.</p> <p>The default is “1”.</p>
[CONFIG]	<i>PING_THREAD</i>	<p>Valid values are 0 (zero) or 1. The default is zero (0).</p> <p>When set to 1, <i>PING_THREAD</i> detects the failure when an Adaptive Server host or network stops running. Internally, the primary companion OpenSwitch checks to see if the entire network has failed or only the Adaptive Server host network. If the Adaptive Server host network has failed, the primary companion OpenSwitch is notified immediately and the action set for <i>SVR_FAIL_ACTION</i> is invoked.</p> <p>When set to zero (0), the primary companion OpenSwitch is notified of a network failure with Adaptive Server after only 8 minutes, which is the default TCP/IP <i>tcp_ip_abort_interval</i> configuration parameter.</p> <hr/> <p>Note Do not use this parameter to monitor the network between clients and OpenSwitch. If the network connection from the client to the OpenSwitch fails, the client detects the failure only when the <i>tcp_ip_abort_interval</i> time has elapsed. This is a kernel parameter that defaults to 8 minutes, but can be tuned to a lower value if the default is unacceptable.</p> <hr/>
[CONFIG]	<i>PING_WAIT</i>	<p>The number of seconds that the ping command should wait before returning a failure. This parameter is used in conjunction with <i>PING_RETRIES</i>, on platforms where the ping command blocks instead of returning right away.</p> <p>The default is 10 (seconds).</p>
[CONFIG]	<i>PRIMARY_COMPANION</i> <hr/> <p>Note Required only when <i>MUTUAL_AWARE</i>=1.</p> <hr/>	<p>Enter:</p> <ul style="list-style-type: none"> • 0 – the default. • 1 – when this is the designated primary companion. A primary companion is responsible for writing to the Adaptive Server cluster tables.

Section	Parameter	Description
[CONFIG]	<p><i>SVR_FAIL_ACTION</i></p> <hr/> <p>Note Required only when <i>MUTUAL_AWARE=1</i>.</p> <hr/>	<p>Enter one of these actions:</p> <ul style="list-style-type: none"> • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL <p>This parameter is used when an Adaptive Server fails to respond in a timely manner, or when the Adaptive Server host cannot be pinged by either OpenSwitch server in a cluster.</p> <p>See “SVR_FAIL_ACTION” on page 122 for more information. See “User-specified actions” on page 113 for details about the available actions.</p>
[CONFIG]	<i>UPDATE_CFG</i>	<p>When set to 1, the configuration file is updated each time a reconfiguration takes place. Zero (0) is the default setting. You must run mutual-aware support with <i>UPDATE_CFG=1</i>.</p>
[SERVER]	<i>CFG_STORAGE</i>	<p>When this property is set for an Adaptive Server, that Adaptive Server is used to store the configuration information of the mutually-aware OpenSwitch clusters. Enter:</p> <ul style="list-style-type: none"> • 0 – do not use this Adaptive Server to store the configuration information of the mutually-aware OpenSwitch clusters. This is the default value. • 1 – store the configuration information of the mutually-aware OpenSwitch clusters on this Adaptive Server. <p>In a mutually-aware setup, you must include this parameter under the two Adaptive Server entries; for example:</p> <pre>[SERVER=ASE1] STATUS=UP CFG_STORAGE=1 [SERVER=ASE2] STATUS=UP CFG_STORAGE=1</pre> <p>In the example above, the most recent OpenSwitch configuration is stored in the <i>syso_<cluster></i> table in ASE1 and ASE2. Therefore, when a mutually-aware OpenSwitch server fails to connect to its companion OpenSwitch for the latest configuration information, it queries both ASE1 and ASE2 to retrieve that information.</p>

Section	Parameter	Description
[COMPANION]	<i>companion_name</i> <i>admin_user</i> <i>admin_password</i>	<p>Name of the OpenSwitch companion, and the administrator user name and password used to make a connection. For example:</p> <pre>[COMPANION=OSW2] admin_user=sa admin_password=sa</pre> <p>These should be the same as the values for <i>SERVER_NAME</i>, <i>ADMIN_USER</i> and <i>ADMIN_PASSWORD</i> in the companion OpenSwitch configuration file.</p> <p><i>companion_name</i> is a static parameter, while <i>admin_user</i> and <i>admin_password</i> are dynamic parameters. This means that after an OpenSwitch server starts running, you can change the companion's <i>admin_user</i> and <i>admin_password</i> using <i>rp_cfg</i> and the changes take effect immediately. However, if you change <i>companion_name</i>, you must restart the OpenSwitch server for the change to take effect.</p> <hr/> <p>Note If <i>USERNAME_PASSWORD_ENCRYPTED</i> is set to 1, <i>admin_user</i> and <i>admin_password</i> should contain the encrypted string (as should <i>ADMIN_USER</i> and <i>ADMIN_PASSWORD</i> in the OpenSwitch configuration file). Use the -E or -p command line options for encryption. See “Using encrypted user names and passwords” on page 48.</p> <hr/>

Mutually-aware configuration table

When you configure an OpenSwitch to be mutually aware, a configuration table is created and saved to both Adaptive Servers, which eliminates any single point of failure created by one Adaptive Server. This table can be accessed by both the primary and secondary OpenSwitch servers.

The primary OpenSwitch server is responsible for updating the configuration table on the Adaptive Servers.

Note When you configure OpenSwitch by manually editing the configuration file, the mutually-aware configuration table is created the first time you start the OpenSwitch server.

Table 5-2 on page 105 lists the columns in the mutually-aware configuration table created in the Adaptive Servers:

Table 5-2: Mutually-aware configuration table

Column	Description
Pool	Name of the Adaptive Server connection pool. Each pool corresponds to exactly one row.
Pool_status	Status of this pool in the OpenSwitch configuration.
Pool_mode	Select: <ul style="list-style-type: none"> Chained – to have all connections routed to the first server defined within the pool, and have administrative switch requests or automatic failovers send all connections to the next server in the pool. Balanced – to have all servers in a pool used simultaneously (load-balanced) until a server fails, at which time all connections on the failed server are redistributed, in round-robin fashion, among the remaining servers.
Pool_cache	The number of seconds a connection is kept alive after a client disconnects. If the same client, using the same user name and password, reconnects during this duration, the connection can be handed off without establishing a new connection, which reduces the overhead of establishing connections each time a client connects.
Attr_username	User name attribute for this pool, if any.
Attr_appname	Application name attribute for this pool, if any.
Attr_hostname	Host name attribute for this pool, if any.
Attr_type	Whether this pool handles “site” or “client” connections.
Primary_server	Name of the primary server in this pool.
Primary_status	The status of the primary server in the OpenSwitch configuration. Primary_status is the same as the server status in the [SERVER] section of the OpenSwitch configuration file, which may be different from the pool-specific server status under the [POOL] section. A mutually-aware OpenSwitch server does not currently support pool-based server status.
Primary_type	Whether the primary server is a high-availability server.
Primary_cfgstore	Whether CFG_STORAGE is set for the primary server.
Primary_cmonuser	The primary server-specific CMON user name.
Primary_cmonpwd	The primary server-specific CMON password.
Secondary_server	Name of the secondary server in this pool.
Secondary_status	The status of the secondary server in the OpenSwitch configuration. Secondary_status is the same as the server status in the [SERVER] section of the OpenSwitch configuration file, which may be different from the pool-specific server status under the [POOL] section. A mutually-aware OpenSwitch server does not currently support pool-based server status.
Secondary_type	Whether the secondary server is a high-availability server.
Secondary_cfgstore	Whether CFG_STORAGE is set for the secondary server.
Secondary_cmonuser	The secondary server-specific CMON user name.
Secondary_cmonpwd	The secondary server-specific CMON user name.

Column	Description
Replication_status	The status of the pool replication direction, as determined by the RCM. Currently displays only “Normal.” Note Reserved for future use.
Timestamp	The timestamp for this entry.
Updated_by	Name of the OpenSwitch server that updated this entry.
Sequence_num	A unique identification number generated by the master OpenSwitch for each update event.

If you use `rp_pool_status` or `rp_server_status` to change the status of a pool or server at runtime for a mutually-aware OpenSwitch, the change is propagated to the companion OpenSwitch server and stored in the Adaptive Server configuration table to ensure maximum redundancy and persistence between OpenSwitch start-ups.

Note Currently, only changes to the pool and server status made using `rp_pool_status` and `rp_server_status` are propagated to the companion and stored in the configuration tables. Runtime changes to other pool or server attributes, and changes to the pool and server status made using `rp_cfg`, are not propagated to the companion and stored in the configuration tables.

Table name

The default configuration table name is `syso_<cluster>`, where *<cluster>* is the value of the `MUTUAL_CLUSTER` parameter set in the OpenSwitch server configuration file. This parameter must be the same for both OpenSwitch servers in a mutually-aware implementation.

The table name begins with “sys” so that the table cannot be replicated. This table is created in the default database for the CMON user in the Adaptive Server. When the default database is not explicitly set for the CMON user, the table is created in the master database.

Note If you want the table created in an Adaptive Server database other than the master database, the administrator should set the CMON user’s default database to a database that is suitable for such use. The administrator must understand the relationship between the mutually-aware OpenSwitch and this table.

When a companion OpenSwitch is not running and a mutually-aware OpenSwitch starts up, the information in the configuration table overrides the information in the OpenSwitch's configuration file. To change this order of precedence, start the mutually-aware OpenSwitch with the "-O" option. See "Requirements" on page 94.

OpenSwitch servers from one cluster write only to the table for that cluster, and as much as possible, a write is carried out in both Adaptive Servers in an atomic fashion to ensure redundancy in the stored configuration information.

Because only one row is entered for each pool, and there can be only two pools, there are only two rows in this table.

Configuration data precedence

Mutually-aware OpenSwitch servers use SQL queries, configuration files, and configuration tables that reside on the Adaptive Servers to communicate with each other and stay in sync about effective pool and server properties.

When a mutually-aware OpenSwitch server (for example, OSW1 with *MUTUAL_AWARE* set to "1") starts up, it takes the following steps to retrieve the latest effective pool and server configuration to use for startup:

- 1 First, the OpenSwitch server that is being started queries its mutually-aware companion to ascertain all server and pool configuration information.
- 2 If the companion OpenSwitch server does not respond, the OpenSwitch server queries the Adaptive Server mutually-aware configuration table for the same server and pool configuration information.
 - If only one Adaptive Server responds, and the query was successful, the primary OpenSwitch server uses the query results to start the OpenSwitch server.
 - If both Adaptive Servers respond, OpenSwitch compares the query results from both of them, and uses the entry with the latest sequence number to start.
- 3 If the Adaptive Servers do not respond, the OpenSwitch server starts using its own configuration file.

- 4 After all configuration information has been updated and the OpenSwitch server starts, OpenSwitch tries to update its companion OpenSwitch server, and the Adaptive Servers that can be reached, with the latest information. For each Adaptive Server defined with *CFG_STORAGE*, OpenSwitch spawns a thread to monitor the health of the Adaptive Server and update the Adaptive Server if it comes online.
- 5 To ensure that both Adaptive Servers have the same data in their cluster table, each time an Adaptive Server is updated (including through a timer), the update is also carried out in the other Adaptive Server, as long as the other Adaptive Server is running and reachable.
- 6 If the cluster table does not exist on a *CFG_STORAGE* Adaptive Server, OpenSwitch creates the cluster table and populates it with the current information the first time it starts with the *CFG_STORAGE* Adaptive Server.

When the system administrator or an RCM executes *rp_stop*, *rp_start*, or *rp_switch* on a mutually-aware OpenSwitch server for an entire pool or Adaptive Server, the commands are propagated to the companion OpenSwitch server to be executed on that server as well. If the command fails on the companion OpenSwitch server, an error message is logged.

When the system administrator or an RCM executes *rp_server_status* or *rp_pool_status* to SET the status for a pool or Adaptive Server, the mutually-aware OpenSwitch server repeats the command on the companion OpenSwitch server, and records the change in the OpenSwitch configuration table on each *CFG_STORAGE* Adaptive Server.

If the preceding steps fail, the operation is rolled back and the status of the pool or Adaptive Server is returned to its original state.

Note Commands such as *rp_pool_{add | rem} attrib*, *rp_pool_{add | rem} server*, and *rp_pool_cache* cannot be propagated to the companion OpenSwitch.

OpenSwitch mutually-aware operations

When an OpenSwitch server starts, it sends an RPC to the companion OpenSwitch server to inform the companion that it is running. When the companion OpenSwitch server receives this information, it marks the starting companion server as running, and thereafter communicates with the companion server whenever the pool or server status changes.

To override saved configuration information and start OpenSwitch with the current configuration file, use the “-O” option. See “Starting and stopping OpenSwitch on UNIX” on page 43 or “Starting and stopping OpenSwitch on Windows” on page 44 for details.

When an OpenSwitch server fails in a mutually-aware configuration, the companion OpenSwitch server detects the failure and marks the failed server as “down.” If the failed OpenSwitch server is a primary companion, the secondary OpenSwitch server assumes the role of the primary OpenSwitch server. As the primary, this OpenSwitch server updates the Adaptive Server mutually-aware configuration table.

Active Adaptive Server failover

When an Adaptive Server fails, the OpenSwitch configuration tries to reconnect to the Adaptive Server several times before initiating failover. The following checks are performed to ensure that a failover is really necessary:

- 1 First, the OpenSwitch server that detected the failure tries to connect to the primary Adaptive Server using the CMON user name and password. If the connection succeeds, the failure is treated as an isolated or client-specific incident, and no server-wide failover is performed. The threads that encounter the failure are terminated, and all future incoming clients are directed to the same primary Adaptive Server.
- 2 If step 1 fails, but the host of the first OpenSwitch can still communicate with the host of the primary Adaptive Server, the primary Adaptive Server is assumed to have stopped responding, and the user-specified behavior for the `SVR_FAIL_ACTION` parameter in the configuration file is performed.
- 3 If step 1 fails because the host of the first OpenSwitch cannot communicate with the host of the primary Adaptive Server, the first OpenSwitch checks with the companion OpenSwitch to see if the latter also has trouble communicating with the primary Adaptive Server host.

- 4 If step 3 succeeds, and the companion OpenSwitch host has no problem communicating with the primary Adaptive Server host, the local network of the first OpenSwitch becomes a suspect, and the user-specified behavior for the *NET_FAIL_ACTION* parameter in the configuration file is performed on the first OpenSwitch, which allows its clients to fail over to its companion OpenSwitch. The clients must reconnect, and are directed to the companion OpenSwitch via the Client-Library failover feature.

However, if the companion OpenSwitch also cannot communicate with the primary Adaptive Server host, a failure at the primary Adaptive Server site or network is assumed, and the user-specified behavior for the *SVR_FAIL_ACTION* parameter in the configuration file is performed on the first OpenSwitch.

- 5 If step 3 fails because the first OpenSwitch host cannot communicate with the companion OpenSwitch host, the first OpenSwitch attempts to ping the secondary Adaptive Server host to determine whether its own host has completely gone off the network. If the communication is also broken between the first OpenSwitch and the secondary Adaptive Server host, the first OpenSwitch assumes that it is experiencing a local network failure, and the user-specified behavior for the *NET_FAIL_ACTION* parameter in the configuration file is performed.

However, if communication still exists with the secondary Adaptive Server host, the first OpenSwitch performs the user-specified behavior for the *SVR_FAIL_ACTION* parameter to fail over the clients to the secondary Adaptive Server.

See “Invoking custom and manual scripts” on page 111 for more information about **_FAIL_ACTION* functionality.

Failback

Failback is a manual process that should be performed with careful planning. It may involve manually restarting the RCM (when an RCM is used), and manually reversing the replication direction of the Replication Server:

- 1 Ensure that the primary Adaptive Server is in its normal running state and accepting connections.

- 2 If there is a secondary Adaptive Server running, and the direction of the replication has been switched during the failover, restore the replication direction to its original state so that the primary Adaptive Server is properly updated with the latest transactions. See the *Replication Server Administration Guide* for instructions.
- 3 If you are using an RCM, restart it. When the RCM is started using the OpenSwitch *RCM_AUTOSTART* parameter, you may need to restart the OpenSwitch servers to restart the RCM. See “Configuring an RCM to start automatically from OpenSwitch” on page 55 for details.
- 4 Log in to the mutually-aware OpenSwitch as an administrator, and execute the following commands. Perform this step on only one of the OpenSwitch servers (either the primary or the secondary); the commands are automatically propagated to the other companion OpenSwitch.

```
rp_stop <POOL>, <secondary_ASE>, NULL, 0, 1
rp_server_status <primary_ASE>, UP
rp_switch <POOL>, <secondary_ASE>, NULL,
    <primary_ASE>, 0, 1
rp_start <POOL>, <secondary_ASE> NULL
```

All future incoming clients are redirected back to the primary Adaptive Server.

Invoking custom and manual scripts

This section provides an overview of using custom and manual scripts when a problem or failure occurs in a mutually-aware environment. It also explains how to use each failure type, describes possible actions that you can invoke for each failure type, and discusses the use of reason and exit codes.

Overview

In a mutually-aware setup, problems may occur with an Adaptive Server or companion OpenSwitch (for example, the server stops responding or there is a network failure) that require a system administrator to perform certain actions. These actions are never required at startup, because any or all servers may not be responding at that time.

When a problem occurs after startup, OpenSwitch invokes the action specified in the OpenSwitch configuration file for that failure type. Depending on the failure type and the corresponding user-specified action, either a custom or manual script is launched, or a default action is taken, or both.

Failure types

OpenSwitch provides four failure type configuration parameters for mutually-aware OpenSwitch servers:

- `CMON_FAIL_ACTION` – used when the CMON thread that monitors the health of the Adaptive Server fails to start.

Note `CMON_FAIL_ACTION` is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with `CMON` set to 1.

- `CMP_FAIL_ACTION` – used when a network failure is detected between the local host and the host of the companion OpenSwitch in a mutually-aware setup.
- `NET_FAIL_ACTION` – used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts.
- `SVR_FAIL_ACTION` – used when an Adaptive Server fails to respond in a timely manner, or when the Adaptive Server host cannot be pinged by either companion OpenSwitch server in a cluster.

When you use the OpenSwitch GUI configuration tool to configure mutually-aware behavior, you specify the actions that should occur for each failure type in the Failure Type dialog box, shown in Figure 5-1 on page 113.

When you manually configure OpenSwitch by editing the configuration file, you specify the action that should occur for each failure type parameter (listed above).

Figure 5-1: Mutually-aware failure types and actions

The screenshot displays a configuration window with a light gray background. It contains five distinct sections, each with a title and a label 'Action To Take'. The first four sections—'Connection Monitor Failure', 'Network Failure', 'Companion Failure', and 'Dataserver Failure'—each have a dropdown menu currently set to 'DEFAULT'. The fifth section, 'User Action', contains two text input fields labeled 'Custom Script' and 'Manual Script'.

Failure Type	Action To Take
Connection Monitor Failure	DEFAULT
Network Failure	DEFAULT
Companion Failure	DEFAULT
Dataserver Failure	DEFAULT
User Action	Custom Script: <input type="text"/> Manual Script: <input type="text"/>

User-specified actions

During OpenSwitch configuration, you specify one action for each failure type parameter. This section describes each available action.

When a user-created script executes, OpenSwitch suspends all pools and puts all threads to sleep. OpenSwitch resumes the pools and wakes all threads when the specified action completes execution.

Note Because only one of the companions may execute a script that notifies the administrator or restarts the server, the scripts for both OpenSwitch companions should perform the same action.

For details about each action's specific use with each failure type, refer to "CMON_FAIL_ACTION" on page 119, "CMP_FAIL_ACTION" on page 120, "NET_FAIL_ACTION" on page 121, and "SVR_FAIL_ACTION" on page 122.

DEFAULT

Invokes the default OpenSwitch behavior for the failure type. The activity that occurs depends on the failure type. See "Failure types" on page 112.

CUSTOM

When you specify CUSTOM for a problem parameter, the path of the script is retrieved from the *CUSTOM_SCRIPT* parameter, and the custom script is executed with the appropriate server name and reason code that OpenSwitch passed as arguments. The custom script should process the server name and reason code to determine the nature of the failure and the appropriate solution.

Note When the failure type is *SRV_FAIL_ACTION* or *CMON_FAIL_ACTION*, the first argument passed to the user action script is the server name.

When the failure type is *CMP_FAIL_ACTION*, the first argument passed to the script is the name of the companion OpenSwitch.

When the failure type is *NET_FAIL_ACTION*, the first argument passed to the script is "NET."

The second argument passed to the script is always the reason code.

Based on the specified reason code provided by the user in the custom script, the script can perform the appropriate actions—restart an Adaptive Server that has stopped running, perform operating system-level checks, or notify the system administrator via an e-mail message that a problem has occurred.

When the custom script completes execution, the exit code of the script is returned in the return code argument.

Note When you set *MUTUAL_AWARE* to 1, the scripts for both OpenSwitch companions must contain the same information. This is because during a server failover, if you have set *SVR_FAIL_ACTION* to MANUAL or CUSTOM, only one of the companions executes the script that notifies the administrator or restarts the server.

See “CMP_FAIL_ACTION” on page 120, “SVR_FAIL_ACTION” on page 122, and “NET_FAIL_ACTION” on page 121 for details.

Example The following code is a sample of a custom script.

```
# Sample custom script for OpenSwitch
#!/bin/sh
server=$1
reason=$2

    case "$reason" in
        "1000")
            mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The OpenSwitch CMON thread failed to start because $server is not
running.
EOF
                r=0
                ;;
        "1001")
            mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The OpenSwitch CMON thread failed to start because the maximum
connection on $server has been exceeded.
EOF
                r=0
                ;;
        "1004")
            # Remotely restart the ASE server.
            rsh bluebird /sybase/ASE-15_0/scripts/RUN_$server &
            if ( $? != 0)
            then
                mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The Adaptive Server $server went down, and the effort to restart it
failed.
EOF
                    r=1
                else
                    mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The Adaptive Server $server went down and was successfully
restarted.
EOF
                    r=0
                fi
                sleep 10
                ;;
        "1005")
            mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The Mutually-aware OpenSwitch detected a complete network failure.
Please check the network of the OpenSwitch server hosts.
```

```
EOF
    r=5
#Example of nondefault return code.
    ;;
    "1006")
        mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
The Mutually-aware OpenSwitch has encountered a network failure
with its companion OpenSwitch host. Please check the network
between the two mutually-aware OpenSwitch servers hosts.
EOF
        r=0
        ;;
    *)
        mailx -s "OpenSwitch failure Alert!" osw_dba << EOF
A problem was encountered and an invalid reason code $reason was
received. Please check the OpenSwitch error log for any failure
messages.
EOF
        r=99
        ;;
esac

exit $r
```

This example invokes a custom user script, specified by the *CUSTOM_SCRIPT* parameter in the OpenSwitch configuration file, that performs different tasks depending on the reason code passed into the script.

MANUAL

When you specify *MANUAL*, the path of the manual script is retrieved from the *MANUAL_SCRIPT* parameter in the configuration file and executed. While the script is executed, OpenSwitch locks all pools and prevents new clients from logging in to the server, while existing client are unaffected and continue normal operation. Similar to a custom script, you must pass in arguments to help the script determine the action to take and the object on which to take that action.

- When the failure type is *SRV_FAIL_ACTION* or *CMON_FAIL_ACTION*, the first argument passed to the user action script is the server name.
- When the failure type is *CMP_FAIL_ACTION*, the first argument passed to the script is the name of the companion OpenSwitch.
- When the failure type is *NET_FAIL_ACTION*, the first argument passed to the script is "NET."

- The second argument passed to the script is always the reason code.

OpenSwitch is suspended until the system administrator executes `rp_go`. After the script finishes execution, the exit code of the script is returned in the return code argument.

If an administrator misses the message issued from `MANUAL`, issue `rp_pool_help` to see if the status is `SUSPENDED`. If the status is `SUSPENDED`, fix the problem and issue `rp_go`.

Note When `MUTUAL_AWARE` is set to 1, the scripts for both OpenSwitch companions must contain the same information. This is because during a server failover, if you have set `SVR_FAIL_ACTION` to `MANUAL` or `CUSTOM`, only one of the companions executes the script that notifies the administrator or restarts the server.

CUSTOM_MANUAL

Useful to perform a manual intervention and suspend the entire OpenSwitch if a custom script fails.

When you specify `CUSTOM_MANUAL`, the custom script is executed first. The manual script is executed only if the custom script fails with an exit code nonzero.

Reason codes

When you specify `CUSTOM`, `MANUAL`, or `CUSTOM_MANUAL` for a failure type parameter, a user-created custom or manual script is executed with a reason code specific to that failure type. Table 5-3 on page 118 lists the reason codes associated with each failure type.

Table 5-3: Custom and manual script reason codes

Code	Failure type	Description
1000	<i>CMON_FAIL_ACTION</i>	The Adaptive Server cannot be connected to because it is not running.
1001	<i>CMON_FAIL_ACTION</i>	The number of connections to the Adaptive Server has exceeded the maximum user connections limit.
1004	<i>SVR_FAIL_ACTION</i>	A primary Adaptive Server failed, and a restart or failover is necessary.
1005	<i>NET_FAIL_ACTION</i>	The network of the local OpenSwitch host is disabled.
1006	<i>CMP_FAIL_ACTION</i>	The companion OpenSwitch stops unexpectedly.
2000	<i>CMON_FAIL_ACTION</i>	The user name or password for the CMON user is invalid on this Adaptive Server.
3000	<i>CMON_FAIL_ACTION</i>	An unknown error was encountered while starting the CMON thread. Check the Adaptive Server to make sure that it is accepting connections.

Exit codes

When you use custom or manual scripts, your script must exit with a valid exit code. Generally, exit codes are a way to notify an administrator whether or not the script ran successfully.

The section on each failure type contains the exit codes specific to using custom or manual scripts with that type of failure.

See “*CMON_FAIL_ACTION*” on page 119, “*CMP_FAIL_ACTION*” on page 120, “*NET_FAIL_ACTION*” on page 121, and “*SVR_FAIL_ACTION*” on page 122 for details.

When you create a script for execution with **CUSTOM**, **MANUAL**, or **CUSTOM_MANUAL** actions, the script must use the exit command to complete, not use any exit command switches, and provide a valid exit code:

```
exit <Any valid exit code>
```

The exit code you provide dictates what OpenSwitch should do after the script executes. The following sample input would be allowed in a custom or manual-invoked script. The sample uses the exit command, does not use any exit command switches, and uses a valid exit code.

```
@echo off
```

```
...
...
exit 3
```

However, this sample input would not be allowed because it uses the “/B” exit command switch:

```
@echo off
...
...
exit/B 3
```

Warning! Never use an exit code of -1. OpenSwitch uses this value internally to determine the reason for a failure. If a script returns -1, it may interfere with this diagnosis and cause the wrong error to be logged.

The remainder of this document describes each failure type parameter in detail, including the behavior invoked by each action for that parameter, and the reason and exit codes specific to that failure type and action.

CMON_FAIL_ACTION

This parameter is used only when the *CMON* configuration parameter is set to 1 (the recommended setting), and when the *CMON* thread that monitors the health of the Adaptive Server fails to start. See “[CONFIG]” on page 64 for details about the *CMON* parameter.

Note *CMON_FAIL_ACTION* is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with *CMON* set to 1.

Actions and reason
codes

Use:

- **DEFAULT** – to try and restart the *CMON* connection. If the problem has been resolved, OpenSwitch continues with its normal activity. If the problem is unresolved, OpenSwitch fails the client over to the next server.
- **CUSTOM**, **MANUAL**, or **CUSTOM_MANUAL** – to execute a user-specified custom or manual script with a reason code of 1000, 1001, 2000, or 3000 (see Table 5-3 on page 118). OpenSwitch is suspended and waits indefinitely until the system administrator executes `rp_go`.

See “User-specified actions” on page 113 for additional details about these actions.

Exit codes for custom and manual scripts

Use an exit code of 0 (zero). For *CMON_FAIL_ACTION*, the exit code only provides a way to notify the administrator how the script ran. Regardless of the exit code you provide, *CMON_FAIL_ACTION* performs the DEFAULT action after the script executes.

CMP_FAIL_ACTION

Actions and reason codes

This parameter is used when a network failure is detected between the local host and the host of the companion OpenSwitch in a mutually-aware setup.

Use:

- **DEFAULT** – to check whether *FREEZE_CFG_ON_FAIL* is set. When *FREEZE_CFG_ON_FAIL* is set, all future configuration changes for pool or server status are prohibited until the connection to the companion OpenSwitch is restored.

If *FREEZE_CFG_ON_FAIL* is not set, no action occurs. See “[CONFIG]” on page 64 for details about this parameter.

- **CUSTOM, MANUAL, or CUSTOM_MANUAL** – to execute the specified custom or manual script with the reason code 1006. OpenSwitch is suspended and waits indefinitely until the system administrator executes *rp_go*.

Note If *CMP_FAIL_ACTION* is set to **CUSTOM** or **MANUAL**, the respective script that is executed must reside on a local file system. This is because the script executes when the network is not responding and access to remote mounted file systems may no longer exist.

See “User-specified actions” on page 113 for additional details about these actions.

Exit codes for custom and manual scripts

Valid exit codes are:

- **0 (zero)** – the default exit code. If the script exits with a zero (0) status, OpenSwitch performs the DEFAULT action (*CMP_FAIL_ACTION=DEFAULT*).
- **1** – if the script exits with a code of 1, no further action is performed.

NET_FAIL_ACTION

This parameter is used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts. However, the specified action is invoked only when each ping has repeatedly failed more than the number of *PING_RETRIES*, while waiting for a response for longer than the amount of time of *PING_WAIT*.

Actions and reason types

Use:

- **DEFAULT** – to clean up and exit, so that existing clients disconnect. When existing clients reconnect, they fail over to the active OpenSwitch companion.
- **CUSTOM, MANUAL, or CUSTOM_MANUAL** – OpenSwitch is suspended and the specified custom or manual script is executed with the reason code 1005. OpenSwitch is suspended and waits indefinitely until the system administrator executes `rp_go`.

Note If *NET_FAIL_ACTION* is set to **CUSTOM** or **MANUAL**, the respective script that is executed must reside on a local file system. This is because the script executes when the network is not responding and access to remote mounted file systems may no longer exist.

See “User-specified actions” on page 113 for additional details about these actions.

Exit codes for custom and manual scripts

Valid exit codes are:

- 0 (zero) – the default exit code. When the script exits with a zero (0), OpenSwitch performs the default action (`NET_FAIL_ACTION=DEFAULT`).
- 1 – if you do not want the **DEFAULT** action performed, use an exit code of 1 for reason code 1005. For **CUSTOM** and **MANUAL**, when the script exits with 1, OpenSwitch remains up until it can reconnect to the network hosted Adaptive Server or companion OpenSwitch.

When you use **CUSTOM_MANUAL**, the manual script is executed only if the custom script exits with 1 (which indicates failure). When the manual script exits with 1, OpenSwitch remains up until it can reconnect to the network hosted Adaptive Server or companion OpenSwitch.

SVR_FAIL_ACTION

This parameter is used when an Adaptive Server fails to respond in a timely manner, or when the Adaptive Server host cannot be pinged by either OpenSwitch server in a cluster.

Warning! If you are using a CM or RCM, set *SVR_FAIL_ACTION* to DEFAULT. Specifically, you cannot run custom or manual scripts for a server failure when you are using a CM or RCM because each CM and RCM is coded differently and its failover procedure may contradict the actions invoked by a custom or manual script.

When you specify DEFAULT for *SVR_FAIL_ACTION*, OpenSwitch checks whether any CMs or RCMs are connected. When there are CM or RCM connections, OpenSwitch returns, leaving failover actions to be executed by the CM or RCM. When there are no CM or RCM connections, OpenSwitch:

- 1 Marks the primary (failed) Adaptive Server as locked
- 2 Stops all clients on the failed Adaptive Server
- 3 Marks the primary Adaptive Server as DOWN
- 4 Marks the secondary Adaptive Server as UP
- 5 Switches clients from the primary Adaptive Server to the secondary Adaptive Server
- 6 Restarts all clients
- 7 Directs all new connections to the secondary Adaptive Server

Actions and reason codes

Use:

- DEFAULT – to mark the Adaptive Server as not running and initiate a failover process.
- CUSTOM, MANUAL, or CUSTOM_MANUAL – to execute the specified custom or manual script with the reason code 1004, unless you are using a CM or RCM, in which case the action and reason code are ignored, and OpenSwitch allows the CM or RCM to handle the failover.

It is important that the scripts on both OpenSwitch companions perform the same actions because during *SVR_FAIL_ACTION*, only one of the companions executes the script. For example, if the script for OSW1 restarts the server or notifies the administrator, the script for OSW2 should also restart the server or notify the administrator. Although the actions must be the same in both scripts, the commands that invoke those actions can be different; that is, you could use different commands to restart the server as long as the commands produce the same result.

When you specify *MANUAL* or *CUSTOM_MANUAL*, OpenSwitch is suspended and waits indefinitely until the system administrator executes *rp_go*.

See “User-specified actions” on page 113 for additional details about these actions.

Exit codes for custom and manual scripts

Valid exit codes are:

- 0 – the script was successful and OpenSwitch should reconnect all existing clients to the same primary Adaptive Server. This exit code should be returned by a script that has succeeded in restarting the primary Adaptive Server.

OpenSwitch does not change the status of the primary Adaptive Server to *DOWN* and future connections continue to be routed to that Adaptive Server.

- 1 – the script was successful and OpenSwitch should fail over all existing clients to the secondary Adaptive Server. This exit code should be returned by a script that sends a notification about the server error, but does not restart the server that is not responding.

OpenSwitch changes the status of the primary Adaptive Server to *DOWN* and future connections are routed to the next available Adaptive Server in the pool.

- 2 – the script was unsuccessful and OpenSwitch should terminate all existing clients connections. This exit code should be returned if the script has failed and no automatic failover is to be performed.

If the primary Adaptive Server is down, but is restarted before new client connections occur, clients reconnect to the primary Adaptive Server.

If the primary Adaptive Server has not been restarted, new client connections failover to the secondary Adaptive Server and OpenSwitch changes the status of the primary Adaptive Server to *DOWN*.

- 3 – the script was unsuccessful and OpenSwitch should fail over all existing clients to the secondary Adaptive Server. This exit code should be returned if the script has failed, and you want to perform an automatic failover to the next available server.

OpenSwitch changes the status of the primary Adaptive Server to DOWN and all future connections are routed to the next available Adaptive Server in the pool.

Registered Procedures

This chapter describes the registered procedures that you can execute from the command line to perform the switching process, and to monitor and administer user activities. These procedures are provided in addition to the default registered procedures that are built into every Open Server.

Registered procedure	Description
rp_cancel	Cancels processing of a query by one or more clients.
rp_cfg	Causes a given configuration file to be reread while the server is running. Similar to sp_configure.
rp_debug	Turns on and off all debugging options available with the -t command line option.
rp_dump	Dumps connection state information.
rp_go	Resumes all suspended pools after a manual intervention has been requested, and performed by OpenSwitch.
rp_help	Displays every registered procedure in OpenSwitch.
rp_kill	Kills a group of OpenSwitch connections.
rp_msg	Queues a text message to be sent to one or more client connections.
rp_pool_addattrib	Adds an connection attribute name/value pair from a pool.
rp_pool_addserver	Adds a new server to the list of servers within a pool.
rp_pool_cache	Changes or displays the connection caching status for a pool.
rp_pool_create	Creates a new pool of servers.
rp_pool_drop	Drops an existing pool.
rp_pool_help	Displays detailed information about a pool.
rp_pool_remattrib	Removes a connection attribute=value pair to a pool.
rp_pool_remserver	Removes a server from the list of available servers within a pool.
rp_pool_server_status	Displays or sets the status of the server present in the pool. If you use rp_pool_server_status to set the server status for a pool, this value overrides the generic server status set using rp_server_status.
rp_pool_status	Changes the status of a pool.
rp_rcm_connect_primary	Run rp_rcm_connect_primary on a secondary OpenSwitch in a redundant RCM setup, after the primary OpenSwitch is shut down and restarted, to notify the secondary RCM to establish a connection to the primary OpenSwitch.
rp_rcm_list	Displays a list of RCMs known to this OpenSwitch.
rp_rcm_shutdown	Shuts down the specified RCM through OpenSwitch.
rp_rcm_startup	Starts an RCM using the specified path and RCM configuration file.

Registered procedure	Description
rp_replay	Replays SQL statements during failover.
rp_rmon	Displays the contents of all [LIMIT_RESOURCE] sections of the configuration file.
rp_server_help	Displays the name, status, HA-type, configuration storage definition, CMON user name and CMON password for the specified server.
rp_server_status	Displays or changes the status of a remote server. This is a generic server status across all pools, for pools that do not have a pool-specific server status defined.
rp_set	Changes a configuration value.
rp_set_srv	Sets server name for a connection that is requesting a server name from a coordination module.
rp_showquery	Displays query being executed by a <i>spid</i> . <i>rp_showquery</i> only works when OpenSwitch is not in FULL_PASSTHRU mode.
rp_shutdown	Shuts down an OpenSwitch server.
rp_start	Starts a group of connections that were previously stopped with <i>rp_stop</i> .
rp_stop	Stops a group of connections.
rp_switch	Switches one or more connections to another server.
rp_traceflag	Enables or disables SRV_TRACE flags for debugging messages.
rp_version	Displays OpenSwitch version number.
rp_who	Displays information about the current set of user connections. Similar to <i>sp_who</i> .

See Chapter 2, “Coordination Module Routines and Registered Procedures,” in the *OpenSwitch Coordination Module Reference Manual* for a list of registered procedures that you can issue from a coordination module.

Invoking registered procedures

Registered procedures are a form of stored procedure that are built into OpenSwitch, rather than being implemented in Adaptive Server. You can invoke registered procedures within OpenSwitch either:

- As part of an RPC through an Adaptive Server, or
- Directly, as a SQL language command.

Remote procedure call invocation

To execute a registered procedure within OpenSwitch through an Adaptive Server remote procedure call, Adaptive Server must first be informed of the OpenSwitch server's existence. Issue the following, where *OPENSWITCH_NAME* is the name of the OpenSwitch server in the interfaces file being used by the Adaptive Server:

```
sp_addserver OPENSWITCH_NAME
```

After this has been accomplished, OpenSwitch registered procedures can be invoked by connecting directly to the Adaptive Server using *isql* or something similar, and performing:

```
1> exec OPENSWITCH_NAME...rp_who
2> go
```

You can find details on setting up and invoking remote procedures in the *Adaptive Server Enterprise System Administration Guide*.

Direct invocation

Any user who is connected using the name *ADMIN_USER* and password *ADMIN_PASSWORD*, as defined in the configuration file, can invoke registered procedures directly through OpenSwitch's built-in RPC parser. OpenSwitch attempts to capture all language commands and parse them as if they were RPC calls.

RPC calls executed in this fashion must be formatted like this:

```
[EXEC[UTE]] rp_name [value[, value...]]
```

Where:

- *rp_name* – is the registered procedure to be executed
- *value* – is the value of the parameter

Note You cannot pass parameters by name.

rp_cancel

Description	Cancels processing of a client connection.
Syntax	<code>rp_cancel [pool_name, srv_name, spid, why]</code>
Parameters	<p><i>pool_name</i></p> <p>The name of the pool in which the connections should be canceled.</p> <p>Supplying only this argument causes all connections within <i>pool_name</i> to be canceled.</p> <p><i>srv_name</i></p> <p>Cancels connections to remote server <i>srv_name</i>. Supplying only this argument cancels all connections to <i>srv_name</i>.</p> <p><i>spid</i></p> <p>Cancels the connection identified within OpenSwitch by <i>spid</i>. If this parameter is specified while <i>pool_name</i> and <i>srv_name</i> are both NULL, OpenSwitch cancels the <i>spid</i> as specified by this parameter.</p> <p>If either <i>pool_name</i> or <i>srv_name</i> are not NULL, those parameters are verified against the <i>actual pool_name</i> or <i>srv_name</i> of the specified <i>spid</i>. The connection for the specified <i>spid</i> is cancelled only if the supplied <i>pool_name</i> and <i>srv_name</i> exactly match the <i>actual _pool_name</i> and <i>srv_name</i> that the specified <i>spid</i> is connected to. Otherwise, no connection is cancelled.</p> <p><i>why</i></p> <p>Message to be sent to the user of a canceled query. If you do not supply a message, the default is: “Your query was canceled by administrative request (spid #n)”.</p>
Examples	<p>Example 1 Cancels the OpenSwitch connection represented by spid 8.</p> <pre>1> rp_cancel NULL, NULL, 8 2> go</pre> <p>Example 2 Cancels all connections currently established to pool POOL_A.</p> <pre>1> rp_cancel "POOL_A", NULL, NULL 2> go</pre> <p>Example 3 Cancels all connections currently established to pool POOL_A on server SYB_SERV2, sending the message, “Sorry! Your connection was canceled.” to each user.</p> <pre>1> rp_cancel "POOL_A", "SYB_SERV2", NULL, "Sorry! Your connection was canceled." 2> go</pre>

Usage

- Use this procedure with caution. If you do not include any arguments with `rp_cancel`, all OpenSwitch connections are canceled.
- *spid* refers to the OpenSwitch process ID, not the process ID in the remote Adaptive Server.
- To generate a report on the current connections, execute `rp_who`.

Messages

- Indicates the number of OpenSwitch connections that were canceled:

```
rp_cancel: Canceled n spids.
```

- The pool name you supplied does not exist:

```
rp_cancel: Invalid pool name 'pool_name'.
```

- The server name you supplied does not exist or has not been defined within OpenSwitch:

```
rp_cancel: Invalid server name 'srv_name'.
```

See also

`rp_kill`, `rp_who`

rp_cfg

Description Rereads the OpenSwitch configuration file at runtime.

Syntax `rp_cfg config_file`

Parameters *config_file*
The name of the configuration file to be reread. Passing a file name of NULL, default, an empty string, or simply deleting a previous entry, causes the previously processed configuration file to be reread.

Examples **Example 1** This command causes the configuration file to be processed.

```
1> rp_cfg "OpenSwitch.cfg"
2> go
```

Example 2 This command causes the previously processed configuration file to be reread.

```
1> rp_cfg "default"
2> go
```

Example 3 This command processes the *OpenSwitch.cfg* file in */usr/sybase/config* on the same host as the OpenSwitch server.

```
1> rp_cfg "/usr/sybase/config/OpenSwitch.cfg"
2> go
```

On the Windows platform:

```
rp_cfg "c:\Sybase\OSW\config\OpenSwitch.cfg"
```

This command processes the *OpenSwitch.cfg* file in *c:\sybase\OSW\config* on the same Windows host as the OpenSwitch server.

Usage

- When a new configuration file is processed, the way each section of the configuration file is processed differs, as described in Table 6-1.

Table 6-1: Effects of rp_cfg on existing configuration settings

Section	Behavior
[CONFIG]	As new <i>name=value</i> pairs are processed within this section, the current value of <i>name</i> in OpenSwitch is replaced with <i>value</i> . Also, some variables, such as <i>RMON</i> , are meaningful only during start-up, so changing the value of these variables at runtime has no effect (for example, setting <i>RMON</i> to zero (0) does not cause the resource governor to be shut down after OpenSwitch has been started).
[SERVER]	All existing server information is replaced with the contents of the new configuration file.
[COMPANION]	The <i>admin_user</i> and <i>admin_password</i> is replaced with the values from the new configuration file if the companion server name remains the same. If you specify a different companion server name, restart OpenSwitch instead of reconfiguring it using <code>rp_cfg</code> .

Section	Behavior
[POOL]	All existing pool information is replaced with the contents of the new configuration file.
[LIMIT_RESOURCE]	<p>When the new configuration file is processed, all existing [LIMIT_RESOURCE] settings are cleared and replaced by the contents of this section.</p> <ul style="list-style-type: none">After verifying that the specified <i>config_file</i> exists, <i>rp_cfg</i> clears all [LIMIT_RESOURCE] settings before processing the contents. Therefore, if this processing stops before completion, due to a syntax error, these settings may be only partially available within OpenSwitch, and the configuration file must be corrected and reprocessed.<i>rp_cfg</i> removes all available pools in OpenSwitch before it processes the new configuration file. Therefore, there is a small window where no pools are available, and if a client connects at that time, it is disconnected. However, this does not affect existing connections, and since <i>rp_cfg</i> executes quickly and the window is almost negligible. If this causes concern, run <i>rp_cfg</i> when there are fewer clients connecting to OpenSwitch.Do not use <i>rp_cfg</i> to reconfigure OpenSwitch; use another registered procedure call instead. For example, to change a [CONFIG] parameter, use <i>rp_set</i> (see <i>rp_set</i> on page 185); to add a server from to pool, use <i>rp_pool_addserver</i> (see <i>rp_pool_addserver</i> on page 147); to remove a server from a pool, use <i>rp_pool_remserver</i> (see <i>rp_pool_remserver</i> on page 162).Do not use <i>rp_cfg</i> to change the following configuration parameters. Because these parameters are read only once when OpenSwitch starts, changing them at runtime has no effect on a running OpenSwitch. To change these parameters, stop and restart OpenSwitch with new values set in the configuration file.

API_CHECK	LOGIN_TIMEOUT	RMON_INTERVAL
CHARSET	MAX_LOGSIZE	SEC_PRINCIPAL
CMON	MAX_PACKETSIZE	SERVER_NAME
CONNECTIONS	MSGQ_SIZE	SRV_TRACE
CTX_TRACE	MUTUAL_AWARE	STACKSIZE
DEBUG_FILE	MUTUAL_CLUSTER	TRUNCATE_LOG
ECHO_LOG	PING_THREAD	UPDATE_CFG
INTERFACES	RESPONSE_TIMEOUT	USE_DONEINPROCS
LOG_FILE	RMON	USERNAME_PASSWORD_ENCRYPTED

Messages

- rp_cfg was run with an argument of default or "" and no previous configuration file is available:

```
rp_cfg: No default configuration is available.
```

For example, this message appears if OpenSwitch was started without a configuration file.

- This message usually indicates that a syntax error was encountered while processing the configuration file:

```
rp_cfg: Error while processing 'config_file'. See  
error logs.
```

You can find detailed information in the OpenSwitch error logs.

- The configuration file was successfully processed, and new settings have taken effect:

```
rp_cfg: Successfully processed configuration file  
'config_file'.
```

rp_debug

Description	Enables or disables OpenSwitch debugging messages.
Syntax	<code>rp_debug [options, [on off]]</code>
Parameters	<p><i>options</i></p> <p>A list of one or more single-character option flags. Each flag is a toggle; supplying it once enables the option, supplying it again disables the option. Passing an option of "" lists the debugging flags that are currently enabled.</p> <p>Table 6-2 shows the valid debugging options. These are identical to the options you can use with the -t flag at the command line.</p>

Table 6-2: Valid options values

Value	Description
a	Enables all possible debugging flags.
b	Displays attempts to set or test configuration options as described in the configuration file.
c	Displays information about result handling of client-side cursors.
C	Logs interactions between a mutually-aware OpenSwitch, its companion OpenSwitch, and Adaptive Servers.
d	Logs access to data items attached to each thread's user data.
D	Displays information about the handling of dynamic SQL statements.
e	Logs all error messages passing through the OpenSwitch error handlers, even those that are normally suppressed.
f	Shows connection progress information when OpenSwitch is interacting with the coordination module.
F	Display messages related to a coordination module (CM).
g	Displays operations involving security negotiations.
h	Displays messages when entering each event handler.
i	Displays progress information concerning the switching process during a call to <code>rp_switch</code> , such as success or failure of each switch, and which connections fail to go idle within the specified period of time.
j	Shows the connection caching activity.
k	Displays activity of the timer thread (the thread that is responsible for calling timed callbacks within OpenSwitch).
l	Dumps every SQL statement issued through the <code>SRV_LANGUAGE</code> event handler to <i>log_file</i> .
m	Displays every memory allocation and de-allocation (more extensive information may be available at compile time).
n	Displays receipt and handling of cancel or attention requests from client connections.
o	Displays a message each time a command line option value is set or tested.
p	Displays manipulation, use, and assignments of server pools.
q	Displays information about the connection monitor activity.
r	Displays current state and actions of the internal resource monitoring thread.
R	Logs interactions between an OpenSwitch and replication coordination modules (RCMs).
s	Shows access and release of shared and exclusive internal locks (used to prevent concurrent access to internal data structures).

Value	Description
S	Displays the SQL statement that is being executed as part of <code>rp_replay</code> calls.
t	Displays activities of the timer thread that is responsible for periodically waking other sleeping threads.
u	Displays information about result sets being returned to client threads.
U	Logs the user action, such as CUSTOM or MANUAL script execution during a companion OpenSwitch or Adaptive Server failure.
v	Logs ping operations and responses from remote machines.
x	Displays mutex accesses (more detailed view on shared locks).

on

Turns on debugging options, which causes debugging messages to be dumped to the error log file.

off

Turns off debugging options.

Examples

Example 1 Displays all debugging flags and their current state.

```
1> rp_debug
2> go
```

Returns:

flag	description	state
----	-----	-----
b	Attribute set/test	off
c	Client cursor handling	off
C	Mutual Aware	on
d	Thread data access requests	off
D	Dynamic SQL handling	off
e	Error handler calls	on
f	Coordination Module info	on
F	Full Passthru mode	off
g	Open Client/Server security	off
h	Open Server handler calls	off
i	Switching process info	off
j	Connection caching info	off
k	Timer thread state	off
l	Output all language requests	off
m	Memory allocation/deallocation	off
n	Client attention requests	off
o	Command line option set/test	off
p	Pool access requests	off
q	Connection monitor (CMON) thread	off

```

r      Resource monitoring (RMON) thread  off
R      RCM start thread                  off
s      Shared lock acquisition            off
S      SQL Statements to be replayed      off
t      Timed sleep thread                 off
u      Result set handling                 off
U      User Action                        off
v      Network ping thread                off
x      Internal mutex grab/release         off
(28 rows affected)
```

Example 2 Causes all switching information and all attention requests to be reported in the OpenSwitch log file.

```

1> rp_debug "in"
2> go
```

Returns:

flag	description	status
----	-----	-----
i	Switching progress info	Off
n	Client attention requests	On

Usage

- Debugging messages are intended primarily for use by individuals testing functionality, and are not intended for day-to-day use.
- Many of the debugging options dump large amount of information to the log file and may significantly impact OpenSwitch performance.

Messages

Indicates that an invalid option was supplied.

```
rp_debug: Invalid debug flag 'flag'.
```

For valid options, see Table 6-2 on page 134.

rp_dump

Description	Dumps connection state information.
Syntax	<code>rp_dump [[thread mutex all]]</code>
Parameters	<p>thread</p> <p>Dumps information about all user connection threads in OpenSwitch.</p> <p>mutex</p> <p>Dumps information about all mutexes that OpenSwitch owns.</p> <p>all</p> <p>Dumps information about all OpenSwitch threads and mutexes. This is the default if no parameters are specified.</p>
Examples	<pre>1> rp_dump thread 2> go</pre>

Returns:

```
***** THREAD DUMP *****
<spid #15 system pid 21337 state=<NONE> coord=<NONE>>
  server mask=0x0, busy time='11/06/05 21:53:09',
  transtate=CS_TRAN_UNDEFINED,
  app='isql', user='sa', host='loka', db='master',
  conn=0xee12ac8, current='ase2', next='ase2', pool='POOL1',
  proc=0x4095b4, cap set=CS_FALSE, next cursor=0, reason code=0,
  reason text='', function=''
(return status = 0)
```

Usage	<ul style="list-style-type: none"> This procedure is provided only for debugging purposes. The output of <code>rp_dump</code> resembles a more detailed output of <code>rp_who</code>.
-------	--

rp_go

Description	Restarts all pools that were suspended by a MANUAL action or script.
Syntax	<code>rp_go</code>
Parameters	None

Usage

- Issued during an Adaptive Server or companion OpenSwitch failure, after a **MANUAL** intervention has been requested and performed by the administrator. Failure types are *CMON_FAIL_ACTION*, *CMP_FAIL_ACTION*, *NET_FAIL_ACTION*, *SVR_FAIL_ACTION*, or when a mutually-aware OpenSwitch detects that its companion has a different set of mutually-aware configuration parameters, and suspends the pools as a result of it.
- Returns all pools to their original states, which allows all pending clients applications that are running to login.
- *rp_go* has no effect on an OpenSwitch server that is not running.
- If you add a new pool during a **MANUAL** intervention, *rp_go* adds the new pool to the existing pool list.
- If a problem is not resolved by the time *rp_go* is issued, depending on problem type, occasionally, the **MANUAL** intervention is requested repeatedly until the problem is resolved. In this case, you may need to issue *rp_go* each time a manual intervention is performed.
- If a mutually-aware OpenSwitch detects that its companion has different configuration parameters, both OpenSwitch servers are suspended, and you must issue *rp_go* on both companions to resume the mutually-aware behavior.

See also

“Invoking custom and manual scripts” on page 111.

rp_help

Description Displays complete set of registered procedures, and their respective parameters, recognized by OpenSwitch.

Syntax `rp_help`

Examples

```
1> rp_help
2> go
```

Returns:

Procedure Name	Parameters
-----	-----
help	NULL
np_switch_end	NULL
np_switch_start	NULL
rp_cancel	@pool_name, @srv_name, @spid, @why
rp_cfg	@file_name
rp_debug	@flags, @state
rp_dump	@what, @sendtolog
rp_help	NULL
rp_kill	@pool_name, @srv_name, @spid
rp_go	NULL
rp_msg	@pool_name, @srv_name, @spid, @msg
rp_pool_addattrib	@pool_name, @attrib, @value
rp_pool_addserver	@pool_name, @server, @rel_server, @status, @position
rp_pool_cache	@pool_name, @cache
rp_pool_create	@pool_name, @rel_pool, @position, @status, @mode
rp_pool_drop	@pool_name
rp_pool_help	@pool_name
rp_pool_reattrib	@pool_name, @attrib, @value
rp_pool_remserver	@pool_name, @server
rp_pool_status	@pool_name, @status
rp_pool_server_status	@pool_name, @server, @status
rp_rcm_connect_primary	NULL
rp_rcm_list	NULL
rp_rcm_shutdown	@rcm_name

rp_rcm_startup	@rcm_path, @rcm_cfg, @rcm_log, @rcm_retries, @rcm_redundant
rp_rmon	NULL
rp_replay	@spid, @action, @name, @sql, @canfail
rp_server_status	@server_name, @status
rp_server_help	@server_name
rp_set	@parm_name, @parm_value
rp_set_srv	@spid, @server
rp_showquery	@spid
rp_shutdown	NULL
rp_start	@pool_name, @srv_name, @spid
rp_stop	@pool_name, @srv_name, @spid, @ign_tran, @ign_fail
rp_switch	@pool, @srcsrv, @spid, @dstsrv, @grace_period, @force
rp_traceflag	@flags, @state
rp_version	NULL
rp_who	@spid
sp_ps	@spid
sp_serverinfo	@function, @name
sp_who	@spid

(42 rows affected)
(return status = 0)

Usage

rp_help

rp_kill

Description	Shut down a group of OpenSwitch connections.
Syntax	<code>rp_kill [pool_name, srv_name, spid]</code>
Parameters	<p><i>pool_name</i> The name of the pool for which the connections should be shut down. Supplying only this argument causes all connections within <i>pool_name</i> to be shut down.</p> <p><i>srv_name</i> Shuts down connections to remote server <i>srv_name</i>. Supplying only this argument causes all connections to <i>srv_name</i> to be shut down.</p> <p><i>spid</i> Shuts down the connection identified within OpenSwitch by <i>spid</i>. If this argument is specified, <i>pool_name</i> and <i>srv_name</i> are ignored.</p>
Examples	<p>Example 1 Shuts down the OpenSwitch connection represented by <i>spid</i> 8 (as identified by <i>rp_who</i>).</p> <pre>1> rp_kill NULL, NULL, 8 2> go</pre> <p>Example 2 Shuts down all connections currently established to pool POOL_A.</p> <pre>1> rp_kill "POOL_A", NULL, NULL 2> go</pre> <p>Example 3 Shuts down all connections currently established to pool POOL_A on server SYB_SERV2.</p> <pre>1> rp_kill "POOL_A", "SYB_SERV2", NULL 2> go</pre>
Usage	<ul style="list-style-type: none"> • If no arguments are supplied to <i>rp_kill</i>, all connections are killed within OpenSwitch. Use this procedure with caution. • <i>spid</i> refers to the OpenSwitch process ID, not the process ID in the remote Adaptive Server. • To generate a report on the current connections, execute <i>rp_who</i>. • Shutting down a connection causes it to be forcibly removed from OpenSwitch. No messages are delivered to the client.
Messages	<ul style="list-style-type: none"> • Indicates the number of OpenSwitch connections that were killed: <code>rp_kill: Killed <i>n</i> spids.</code>

- The pool name you supplied does not exist:
`rp_kill: Invalid pool name 'pool_name'.`
- The server name you supplied does not exist or has not been defined within OpenSwitch:

`rp_kill: Invalid server name 'srv_name'.`

See also

`rp_cancel`, `rp_who`, `sp_who`

rp_msg

Description	Queues text message to be broadcast to one or more client connections.
Syntax	<code>rp_msg [pool_name], [srv_name], [spid], msg</code>
Parameters	<p><i>pool_name</i> The name of the pool to which the message should be delivered. Supplying only this argument sends the message to all connections in the specified pool.</p> <p><i>srv_name</i> Sends the message to connections currently established to <i>srv_name</i>.</p> <p><i>spid</i> The OpenSwitch process ID of the client connection to receive the message. Connection spid numbers can be obtained using <i>rp_who</i>.</p> <p><i>msg</i> The text of the message to be delivered. This message must be less than 255 characters in length.</p>
Examples	<pre>1> rp_msg POOL1, ase2, 19, "TEST" 2> go</pre> <p>Results:</p> <pre>Successfully queued message to spid 19 (return status = 0)</pre>
Usage	<ul style="list-style-type: none"> • Due to the nature of TDS (the protocol used by clients to communicate with the OpenSwitch server), messages cannot be delivered immediately and, instead, are queued to be sent to clients during the next activity. All clients that were actively processing a result set at the time that <i>rp_msg</i> is issued receive the message at the very end of their result set, and all idle clients receive the message as soon as they initiate a new request of the server. • Idle clients that disconnect without issuing a subsequent query never receive the message. • Do not issue <i>rp_msg</i> from a client connection that is passing through OpenSwitch and using Adaptive Server to invoke the <i>rp_msg</i> RPC in OpenSwitch. This causes the issuing connection to be locked in a transaction, which means that the connection cannot be switched. <p>Messages</p> <ul style="list-style-type: none"> • See the OpenSwitch error log (specified via the <i>-l</i> flag or <i>LOG_FILE</i> configuration option variable) for details:

rp_msg: Error while queuing message. See OpenSwitch error logs.

- The *spid* supplied is not a valid *spid* in OpenSwitch:

rp_msg: *spid* is not a valid spid.

This may be because the *spid* has disconnected since the *rp_msg* request was issued.

- The message has been queued to be delivered to the specified *spid*:

rp_msg: Successfully queued message to spid *spid*.

- This message is displayed when the *spid* argument is -1:

rp_msg: Successfully queued message to N spids.

It indicates the total number of *spids* that have been scheduled to receive the message.

See also

rp_who, sp_who

rp_pool_addattrib

Description Adds connection attribute/value pair to a pool.

Syntax `rp_pool_addattrib pool_name, attrib, value`

Parameters *pool_name*
Identifies the name of the pool to which the attribute/value pair is to be added.

attrib
The name of the attribute to be added to the pool. Table 6-3 shows the valid values.

Table 6-3: Attribute names for *rp_pool_addattrib*

Attribute	Description
username	<i>value</i> must match the user name of an incoming client connection.
appname	<i>value</i> must match the application name identified by the incoming client connection.
hostname	<i>value</i> must match the host machine name identified by the incoming client connection.

value
A standard SQL wildcard expression use to match *attrib*.

Examples **Example 1** Routes any connection created by a user name beginning with “Sybase” or “sybase” to POOL_A.

```
1> rp_pool_addattrib "POOL_A", "username", "[sS]ybase%"
2> go
```

Example 2 Routes any connection created with an application name of “isql” to POOL_A.

```
1> rp_pool_addattrib "POOL_A", "appname", "isql"
2> go
```

Example 3 Routes any incoming connection from this particular host to POOL_A.

```
1> rp_pool_addattrib "POOL_A", "hostname",
   "name of host"
2> go
```

Usage

- An attribute/value pair is used to route connections to a pool. At the time a connection is established to OpenSwitch, the attribute of the connection identified by *attrib* is compared to the regular expression *value*. If a match is found, the connection is routed to *pool_name*.

- Adding or removing attribute/value pairs from a pool has no effect on existing connections; however, the changes apply to existing connections during a failover.
- Both the application name and host name attributes of a client connection must be explicitly set by the client application and cannot actually reflect the real application name and host name that the user is using.
- Changes applied to a pool are not reflected in the configuration file. You must manually change the configuration file.
- Use `rp_pool_help` to display the current set of attributes.

Messages

- The named pool does not exist within OpenSwitch:

```
rp_pool_addattrib: There is no such pool
'pool_name'.
```

Use `rp_pool_create` to create pools. See “`rp_pool_create`” on page 154.

- The attribute named “attrib” is invalid:

```
rp_pool_addattrib: Invalid attribute name 'attrib'.
```

- This message is usually encountered due to an invalid regular expression syntax:

```
rp_pool_addattrib: Error adding attribute 'attrib'
to pool 'pool_name'.
```

Details can be found in the OpenSwitch error log.

- Registered procedure execution succeeded:

```
rp_pool_addattrib: Attribute successfully added to
pool 'pool_name'
```

See also

`rp_pool_create`, `rp_pool_help`, `rp_pool_reattrib`

rp_pool_addserver

Description Adds a remote server name to a pool.

Note `rp_pool_addserver` is not currently support by mutually-aware OpenSwitch servers.

Syntax `rp_pool_addserver pool_name, server [, rel_server, status, position]`

Parameters *pool_name*

Name of the pool to which the server is being added.

server

Name of the remote server being added to the pool.

rel_server

Name of the server within the pool relative to the server being added.

status

The status of the server being added.

- When *status* is NULL, the generic status of this server is used for all pools if it exists in the [SERVER] section of the OpenSwitch configuration file or the status can be retrieved using `rp_server_status`.
- If the server does not exist in the [SERVER] section, OpenSwitch uses the status of the DEFAULT server in the [SERVER] section.
- When *status* is not NULL, the value is used for the server's effective status in this pool, regardless of the generic server status. The status value is used until you change it using `rp_pool_server_status`, or until the server is removed from this pool using `rp_pool_remserver` and re-added with status set to NULL. This is called a pool-specific server status, which facilitates pool-based failover, where individual pools fail over clients to the secondary Adaptive Server even though the primary Adaptive Server is still considered running in other pools.

Once you set a pool's server status using `rp_pool_addserver` or `rp_pool_server_status`, always verify the status using `rp_pool_server_status`, not `rp_server_status`, because the effective pool-specific server status can be different from the generic server status.

Note Pool-specific server status and pool-based failover are not supported by mutually-aware OpenSwitch servers.

Valid *status* values are:

Value	Description
UP	The server is immediately available for use.
DOWN	The server is unavailable, and is not considered for new client connections to OpenSwitch.
LOCKED	The server is available, but new incoming connections being actively connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to client applications to be “stuck” until the pool is unlocked.

position

Valid values are:

Value	Description
HEAD	<i>rel_server</i> is ignored, and <i>server</i> is placed at the beginning of the list of servers in the pool
TAIL	<i>rel_server</i> is ignored, and <i>server</i> is placed at the end of the list of servers in the pool. This is the default position if no value is supplied for <i>position</i> .
BEFORE	<i>server</i> is added immediately before <i>rel_server</i> in the list of servers in the pool.
AFTER	<i>server</i> is added immediately after <i>rel_server</i> in the list of servers in the pool.

Examples

Example 1 Adds server SYB_SERV1 to the end of the list of servers associated with POOL_A with no pool-specific server status.

```
1> rp_pool_addserver "POOL_A", "SYB_SERV1"
2> go
```

Example 2 Adds server SYB_SERV2 with a DOWN status to a position immediately before SYB_SERV1 in the list of servers associated with POOL_A.

```
1> rp_pool_addserver "POOL_A", "SYB_SERV2", "SYB_SERV1", DOWN, "BEFORE"
2> go
```

Example 3 Adds server SYB_SERV3 with status UP as the first server in the list of servers associated with POOL_A.

```
1> rp_pool_addserver "POOL_A", "SYB_SERV3", NULL, UP, "HEAD"
2> go
```

Example 4 Adds server SYB_SERV2 to a position immediately before SYB_SERV1 in the list of servers associated with POOL_A. The status of SYB_SERV2 is the status of the DEFAULT server.

```
1> rp_pool_addserver "POOL_A", "SYB_SERV2", "SYB_SERV1", NULL, "BEFORE"
2> go
```

Example 5 Adds server “ase3” to a position immediately before the server “ase1” in the list of servers associated with pool POOL1.

```
1> rp_pool_addserver "POOL1", "ase3", "ase1", NULL, "BEFORE"
2> go
```

Returns:

```
rp_pool_addserver: Server ase3 added to pool POOL1
```

```
(return status = 0)
```

Afterward, the `rp_pool_help` command is issued to display the information below about POOL1, such as status and mode (chained or balanced).

```
1> rp_pool_help POOL12> go
```

Returns:

```
pool_name    mode      cache    status    block    next_server
-----
POOL1        CHAINED    0        UP        0        ase3
(1 row affected)

server_name
-----
ase3
ase1
ase2
(3 rows affected)

attribute    value
-----
(0 rows affected) (return status = 0)
```

Usage

- When a connection is first established within a pool, it is routed to the first (in chained mode) or next (in balanced mode) available server, therefore, the order in which you define servers in the pool is important.
- To display the current set of servers within a pool, enter:

```
rp_pool_help "pool_name"
```

- At the time a server is added to a pool, OpenSwitch does not validate the name of the server in the interfaces file. Make sure that the server name is accurate.

Messages

- The supplied pool name does not exist in OpenSwitch:

```
rp_pool_addserver: There is no such pool
'pool_name'.
```

You can create a new pool with `rp_pool_create`.

- The server does not exist in the pool:

```
rp_pool_addserver: There is no such server as
'rel_server' in pool 'pool_name'.
```

You can add a server using `rp_pool_addserver`.

- Registered procedure call succeeded:

```
rp_pool_addserver: Server 'server' added to pool  
'pool_name'.
```

See also

`rp_pool_create`, `rp_pool_remattrib`, `rp_pool_server_status`, `rp_pool_remserver`

rp_pool_cache

Description Sets or displays pool cache setting.

Syntax rp_pool_cache [*pool_name*, *cache*]

Parameters *pool_name*

 Name of the pool to be displayed or changed.

cache

 The number of seconds that connection caches are held in the pool. Setting this to a value of zero (0) disables future connection caching.

Examples

Example 1 Displays the list of all pools and their current cache values.

```
1> rp_pool_cache
2> go
```

Returns:

pool_name	cache
POOL_A	0
POOL_B	30
POOL_C	0

Example 2 Displays the current cache value for POOL_A.

```
1> rp_pool_cache "POOL_A"
2> go
```

Returns:

pool_name	cache
POOL_A	0

Example 3 Changes the cache of POOL_A to 10 seconds.

```
1> rp_pool_cache "POOL_A", 102
> go
```

Returns:

pool_name	cache
POOL_A	10

Example 4 Changes the cache of all pools to 30 seconds.

```
1> rp_pool_cache NULL, 30
2> go
```

Returns:

pool_name	cache
-----	-----
POOL_A	30
POOL_B	30
POOL_C	30

Usage

- The cache value for a pool indicates the number of seconds that an outgoing connection is to be maintained following a disconnection from a client application. Connection caching can greatly improve performance of applications that rapidly create short-duration connections.
- If you use connection caching, verify that the client application resets all necessary connection options after every new connection. OpenSwitch does not reset connection options when it reuses a cached connection to the Adaptive Server. Option settings from a previous connection continue to take effect for new client connections that use the same user name and password until you reset the options.
- Changing the cache duration for a pool does not affect those connections that are already cached; it only affects future connections.
- For more details, see “Using connection caching” on page 34.

Messages

- The pool does not exist within OpenSwitch:

```
rp_pool_cache: There is no such pool 'pool_name'.
```

The list of existing pools can be determined using `rp_pool_help`. New pools can be defined using `rp_pool_create`.

See also

`rp_pool_create`, `rp_pool_help`

rp_pool_create

Description Creates a new pool.

Syntax `rp_pool_create pool_name [, rel_pool, position, status, mode]`

Parameters *pool_name*

Name of the pool to be created.

rel_pool

Name of an existing pool, relative to which *pool_name* will be created.

position

Position of *pool_name* relative to *rel_pool*. Table 6-4 describes the values for *position*.

Table 6-4: Position values for rp_pool_create

Position	Description
HEAD	<i>rel_pool</i> is ignored, and <i>pool_name</i> is placed at the beginning of the list of pools.
TAIL	<i>rel_pool</i> is ignored, and <i>pool_name</i> is placed at the end of the list of servers within the pool. This is the default position if position is not supplied.
BEFORE	<i>pool_name</i> is added immediately before <i>rel_pool</i> in the list of pools.
AFTER	<i>pool_name</i> is added immediately after <i>rel_pool</i> in the list of pools.

status

The initial status of the pool. Table 6-5 describes the values for *status*.

Table 6-5: Status values for rp_pool_create

Status	Description
UP	The pool is immediately available for use.
DOWN	The pool is unavailable, and will not be considered for use by any new client connections established to OpenSwitch. This is the default value.
LOCKED	The pool is available, but any new incoming connections actively being connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to the client applications to have stopped responding until the pool is unlocked.

mode

The routing and switching mode of the pool. Table 6-6 on page 155 describes the values for *mode*.

Table 6-6: Mode values for `rp_pool_create`

Mode	Description
CHAINED	All connections are routed to the first server in the pool that has a status of UP or LOCKED.
BALANCED	Incoming connections are routed to servers in round-robin fashion among every server within the pool that has a status of UP or LOCKED, effectively balancing user load across all servers.

Examples

Example 1 Creates chained mode POOL_A at the end of the list of existing pools. This pool has a status of DOWN.

```
1> rp_pool_create "POOL_A"
2> go
```

Example 2 Creates chained mode POOL_A immediately before POOL_B in the list of existing pools. This pool has a status of DOWN.

```
1> rp_pool_create "POOL_A", "POOL_B", "BEFORE"
2> go
```

Example 3 Creates balanced mode POOL_A at the end of the list of existing pools. This pool has a status of LOCKED.

```
1> rp_pool_create "POOL_A", NULL, NULL, "LOCKED",
    "BALANCED"
2> go
```

Example 4 Creates chained mode POOL_C at the end of the list of existing pools. This pool has a status of DOWN.

```
1> rp_pool_create "POOL_C", NULL, TAIL
2> go
```

Example 5 Creates a chained mode POOL_D after POOL_B with status equal to UP.

```
1> rp_pool_create "POOL_D", POOL_B, AFTER, UP, CHAINED
2> go
```

Usage

- The order in which pools are defined is important; all connections are routed to the first matching pool according to the attribute/value pairs established using `rp_pool_addattrib`. See `rp_pool_addattrib` on page 145.
- Pools are initially created with no associated servers or attributes. Therefore, Sybase strongly recommends that you assign pools a status of DOWN. This prevents connections from being routed to the pool until its construction is complete.

- Creating a new pool has no effect on existing connections. However, a new pool is considered when existing connections are being switched (for example, due to failover).
- To specify a position of BEFORE or AFTER, you must also supply a relative pool.

Messages

- The supplied pool name has already been created:

```
rp_pool_create: There is already a pool named  
'pool_name'.
```

Either destroy or modify the existing pool.

- The name of the relative pool supplied does not exist:

```
rp_pool_create: There is no such pool as 'rel_pool'.
```

- A position of BEFORE or AFTER was specified without a corresponding relative pool:

```
rp_pool_create: NULL @rel_pool with 'position'  
position.
```

See also

`rp_pool_addserver`, `rp_pool_addattrib`

rp_pool_drop

Description	Drops an existing pool.
Syntax	<code>rp_pool_drop pool_name</code>
Parameters	<i>pool_name</i> Name of the pool to be dropped.
Examples	<p>Drops the pool named “POOL2” from the pool list in the OpenSwitch server.</p> <pre>1> rp_pool_drop POOL2 2> go</pre> <p>Returns:</p> <pre>rp_pool_drop: Pool POOL2 dropped (return status = 0)</pre>
Usage	<ul style="list-style-type: none">• You must supply the name of the pool to be dropped. The pool name must be exactly the same as what is declared in the <i>config</i> file.• You can use <code>rp_pool_help</code> to get a list of the current pool names. <p>Messages</p> <ul style="list-style-type: none">• You did not supply a pool name for the pool to be dropped: <pre>Procedure rp_pool_drop expects parameter @pool_name, which was not supplied</pre>• The pool with the supplied pool name does not exist: <pre>rp_pool_drop: There is no such pool as 'pool_name'</pre><p>Run <code>rp_pool_status pool_name</code> to check your pool list.</p>
See also	<code>rp_pool_create</code> , <code>rp_pool_status</code>

rp_pool_help

Description Displays information about pools.

Syntax rp_pool_help [pool_name]

Parameters pool_name
 Name of the pool about which information is to be displayed. Without this parameter, rp_pool_help issues a report about all available pools.

Examples **Example 1** Displays information about all pools.

```
1> rp_pool_help
2> go
```

Returns:

pool_name	mode	status	block	next_server
-----	-----	-----	-----	-----
POOL_A	CHAINED	UP	0	SYB_SERV1
POOL_B	BALANCED	UP	0	SYB_SERV2
POOL_C	BALANCED	LOCKED	0	SYB_SERV3

Example 2 Displays information about POOL_A.

```
1> rp_pool_help "POOL_A"
2> go
```

Returns:

pool_name	mode	status	block	next_server
-----	-----	-----	-----	-----
POOL_A	CHAINED	UP	0	SYB_SERV1
(1 row affected)				

server_name

SYB_SERV1
SYB_SERV3
(2 rows affected)

attribute	value
-----	-----
appname	isql
hostname	test.sybase.com

Usage • The block column indicates the number of spids that are current blocked on a LOCKED pool.

- The `next_server` column indicates the name of the next server to be assigned to an incoming connection or a connection in the process of switching.

Messages

- The specified pool does not exist in the OpenSwitch:

```
rp_pool_help: Invalid pool name 'pool_name'.
```

Run `rp_pool_help` without *pool_name* to determine the set of available pools.

See also

`rp_pool_addattrib`, `rp_pool_addserver`, `rp_pool_create`

rp_pool_remattrib

Description Removes a connection attribute/value from a pool.

Syntax `rp_pool_remattrib pool_name, attrib, value`

Parameters

pool_name
Identifies the name of the pool from which the attribute/value pair is to be removed.

attrib
The name of the attribute to be removed from the pool. Table 6-7 describes the values for *attrib*.

Table 6-7: Attribute names for rp_pool_remattrib

Attribute	Description
<i>username</i>	Value must match the user name of an incoming client connection.
<i>appname</i>	Value must match the application name identified by the incoming client connection.
<i>hostname</i>	Value must match the host machine name identified by the incoming client connection.

value
A standard SQL wildcard expression used to match *attrib*.

Examples **Example 1** Removes attribute *appname*, with a value of “isql%” from POOL_A.

```
1> rp_pool_remattrib "POOL_A", "appname", "isql%"
2> go
```

Example 2 Removes the attribute *hostname* with the a value of “[Cc]rater” from POOL_A.

```
1> rp_pool_remattrib "POOL_A", "hostname", "[Cc]rater"
2> go
```

Example 3 Removes the attribute *username* with a value of “Moon” from POOL_A.

```
1> rp_pool_remattrib "POOL_A", "username", "Moon"
2> go
```

- Usage
- The value parameter must exactly match the actual attribute’s value, either as specified in the configuration file, or by using `rp_pool_remattrib` to successfully remove the attribute.
 - Adding or removing *attribute=value* pairs from a pool has no effect on existing connections except during a failover.

- Any changes you make to a pool using `rp_pool_remattrib` are not reflected in the configuration file; you must make those changes manually.
- You can use `rp_pool_help` to display the current attributes.

Messages

- The named pool does not exist within OpenSwitch:

```
rp_pool_remattrib: There is no such pool  
'pool_name'.
```
- The named attribute is not one of user name, host name, or application name:

```
rp_pool_remattrib: Invalid attribute name 'attrib'.
```
- Registered procedure execution succeeded:

```
rp_pool_addattrib: Attribute successfully removed  
from pool 'pool_name'
```

See also

`rp_pool_addattrib`, `rp_pool_create`, `rp_pool_help`

rp_pool_remserver

Description Removes a server from a pool.

Note `rp_pool_remserver` is not currently support by mutually-aware OpenSwitch servers.

Syntax `rp_pool_remserver pool_name, server`

Parameters *pool_name*

The name of the pool from which the server is to be removed.

server

The name of a remote server that belongs to *pool_name*.

Examples Removes SYB_SERV1 from the list of servers available in POOL_A.

```
1> rp_pool_remserver "POOL_A", "SYB_SERV1"
2> go
```

Usage

- Removing a server from a pool has no effect on existing connections using the server; they remain attached to the server. To remove connections from server, use `rp_switch`.
- Leaving a pool with no available servers causes all connections routed to the pool to fail to connect due to lack of available servers.
- You can use `rp_pool_help` to determine the set of servers available within a pool.

Messages

- The pool does not exist within OpenSwitch:

```
rp_pool_remserver: There is no such pool
'pool_name'.
```

- The server does not exist within the pool:

```
rp_pool_remserver: There is no server 'server' in
pool 'pool_name'
```

- The registered procedure executed successfully:

```
rp_pool_remserver: Server 'server' removed from pool
```

See also `rp_pool_addserver`, `rp_pool_help`

rp_pool_server_status

Description Displays or sets the status of the server present in any pool that is defined in the [POOL] section of the OpenSwitch configuration file.

Syntax `rp_pool_server_status pool [server, status]`

Parameters *pool*
The name of the pool. The pool you specify must be defined in the [POOL] section of the OpenSwitch configuration file.

server
Name of the server. If server name is NULL, `rp_pool_server_status` displays the status of all servers present in the pool.

status
The status of the *server*. If status is NULL, then `rp_pool_server_status` displays the status of the specified server that is present in the pool. If *status* is not NULL, the status value is used to set the pool-specific server status for this pool.

Valid *status* values are:

Value	Description
UP	The server is immediately available for use.
DOWN	The server is unavailable, and is not considered for new client connections to OpenSwitch.
LOCKED	The server is available, but new incoming connections being actively connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to client applications to be “stuck” until the pool is unlocked.

Examples

Example 1 Displays the status of all the servers present in the “POOL1.”

```
1> rp_pool_server_status "POOL1", NULL, NULL
2> go
```

Returns:

```
pool_name  server_name  status
-----
POOL1      ase1         UP
POOL1      ase2         UP
POOL1      ase3         UP
```

Example 2 Displays the status of the “ase1” server that is present in “POOL1.”

```
1> rp_pool_server_status "POOL1", "ase1", NULL
```

```
2> go
```

Returns:

pool_name	server_name	status
POOL1	ase1	UP

Example 3 Sets the status to DOWN of the “ase1” server that is present in the “POOL1.”

```
1> rp_pool_server_status "POOL1", "ase1", "DOWN"
2> go
```

Returns:

pool_name	server_name	status
POOL1	ase1	DOWN

Usage

The status set by `rp_pool_server_status`, if any, takes precedence over the status set by `rp_server_status` for a server status within a pool.

- Once a pool is assigned a pool-specific server status, it always uses that server status, instead of the generic server status, to determine where to send incoming client connections. This allows pool-based failover, where two or more pools share the same primary Adaptive Server, some pools channel client connections to the secondary Adaptive Server, and other pools continue to send client connections to the same primary Adaptive Server.
- After you use `rp_pool_server_status` to set a server’s pool status, you can no longer use `rp_server_status` to accurately display the status for that server pool. Use `rp_pool_server_status` to verify the server status instead.
- To remove a pool-specific server status that you set using either `rp_pool_server_status` or `rp_pool_addserver`, first drop the server from the pool using `rp_pool_remserver`, then re-add the server using `rp_pool_addserver` with NULL as the status.
- Do not use `rp_pool_server_status` when *MUTUAL_AWARE* is set to 1. Pool-based failover is not supported by mutually-aware OpenSwitch servers, and a pool-specific server status may not match the generic server status set by the default *SVR_FAIL_ACTION* during a failover.

See also

`rp_pool_addserver`, `rp_server_status`

rp_pool_status

Description	Sets or displays pool status.
Syntax	<code>rp_pool_status [pool_name, status]</code>
Parameters	<p><i>pool_name</i> Name of the pool to be displayed or changed.</p> <p><i>status</i> The status to which the pool is to be changed. If you supply a status value, but no pool name, the status of all pools is changed. Table 6-8 describes the values for <i>status</i>.</p>

Table 6-8: Status values for *rp_pool_status*

Status	Description
PRE_UP	<p>Mutually-aware-specific pool status. The pool is either in the process of being marked as UP, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to UP on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_UP.</p> <hr/>
UP	The pool is immediately available for use.
PRE_DOWN	<p>Mutually-aware specific pool status. The pool is either in the process of being marked as DOWN, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to DOWN on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_DOWN.</p> <hr/>
DOWN	The pool is unavailable, and is not considered for use by any new client connections established to OpenSwitch.
PRE_LOCKED	<p>Mutually-aware specific pool status. The pool is either in the process of being marked as LOCKED, or has encountered a problem during that process. doing so. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the pool status to LOCKED on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a pool's status to PRE_LOCKED.</p> <hr/>

Status	Description
LOCKED	The pool is available, but any new incoming connections actively being connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to the client applications to have stopped responding until the pool is unlocked.
SUSPENDED	The pool is being suspended by OpenSwitch due to a failure that requires an administrator's manual intervention. See "Invoking custom and manual scripts" on page 111 for more information. The pool blocks on all new connections until rp_go is issued.
	<hr/> Warning! Do not manually set a pool's status to SUSPENDED. <hr/>

Examples

Example 1 Displays the list of all pools and their current status:

```
1> rp_pool_status
2> go
```

Returns:

```
pool_name      status
-----
POOL_A         UP
POOL_B         UP
POOL_C         UP
```

Example 2 Displays the current status of POOL_A:

```
1> rp_pool_status "POOL_A"
2> go
```

Returns:

```
pool_name      status
-----
POOL_A         UP
```

Example 3 Changes the status of POOL_A to LOCKED and displays the results:

```
1> rp_pool_status "POOL_A", "LOCKED"
2> go
```

Returns:

```
pool_name      status
-----
POOL_A         LOCKED
```

Usage

- Changing the status of a pool does not affect users that are currently using the pool.
- Changing the status of a pool has no affect on existing connections.
- If you do not have a pool-based server, use `rp_server_status` to set and check a server's status.

If you have a pool-based server, use `rp_pool_server_status` to set and verify the server's status.

- `rp_server_status` displays the status of a server only if that server is listed in the [SERVER] section of the OpenSwitch configuration file, which may not be the same as a pool's actual server status.
- Connections that are currently blocked on a LOCKED pool continue to remain blocked until either the pool is unlocked or the client application disconnects. This means that any administrative requests made of the connection, such as a call to `rp_switch`, or `rp_stop`, are queued until the pool changes status.
- Use `rp_pool_status` with the LOCKED argument, followed by a call to `rp_stop`, to display all activity on a pool.
- If you issue `rp_pool_status` to set the pool status on a mutually-aware OpenSwitch server, the command is propagated to the companion OpenSwitch if it is running. The new pool status is also recorded in the mutually-aware configuration tables on the Adaptive Servers. If these steps fail, the pool status is reset to its original value and an error message is logged.
- If FREEZE_CFG_ON_FAIL is enabled and the network fails between the local OpenSwitch server and the companion OpenSwitch server, you can use `rp_pool_status` only to display, not set, a pool's status. This prevents the companion OpenSwitch servers from switching client connections to different Adaptive Servers while the network between OpenSwitch servers is not responding, which could cause data loss when replication is performed in only one direction. `rp_pool_status` allows you to reset the pool status when the network is restored between the companion OpenSwitch servers, or if FREEZE_CFG_ON_FAIL is disabled.

Messages

- The pool name does not exist within OpenSwitch:

`rp_pool_status: There is no such pool 'pool_name'.`

To list the existing pools, use `rp_pool_help`.

- A mutually-aware OpenSwitch server has detected a problem in the network with its companion, and has stopped all future status changes until the network is restored and the status of the companion can be verified:

```
rp_pool_status:Status cannot be set/changed until  
connectivity is restored with the companion  
OpenSwitch site or the FREEZE_CFG_ON_FAIL parameter  
is turned OFF.
```

See also

`rp_pool_create`, `rp_pool_help`

rp_rcm_connect_primary

Description	Instructs the secondary RCM to establish a connection to the primary OpenSwitch after it restarts.
Syntax	<code>rp_rcm_connect_primary</code>
Parameters	None.
Usage	In a redundant RCM environment, issue <code>rp_rcm_connect_primary</code> on the secondary OpenSwitch after the primary OpenSwitch has gone down and been restarted.
See also	<code>rp_rcm_list</code>

rp_rcm_list

Description	Displays a list of RCMs known to the OpenSwitch on which <code>rp_rcm_list</code> is executed. The list displays the RCM name and whether the RCM is a primary or secondary RCM.
Syntax	<code>rp_rcm_list</code>
Parameters	None.
Usage	<ul style="list-style-type: none">• Used to check which RCMs are connected to an OpenSwitch.• You can also use <code>rp_rcm_list</code> to generate the name of the RCM to shut down through OpenSwitch using <code>rp_rcm_shutdown</code>.
See also	<code>rp_rcm_shutdown</code> , <code>rp_rcm_connect_primary</code> , <code>rp_rcm_startup</code>

rp_rcm_shutdown

Description	Shuts down the named RCM through OpenSwitch.
Syntax	<code>rp_rcm_shutdown rcm_name</code>
Parameters	<p><i>rcm_name</i></p> <p>The name of the RCM specified for the <i>RCM_NAME</i> parameter in the RCM configuration file.</p> <ul style="list-style-type: none">• If you do not specify <i>rcm_name</i>, this parameter's value defaults to "<i><OPENSWITCH>_rcm</i>," where <i><OPENSWITCH></i> is another parameter set in the RCM configuration file.• If <i>RCMNAME</i> is not set in the RCM configuration file, enter "<i><OPENSWITCH>_rcm</i>" for <i>rcm_name</i> to shut down the RCM that was started by this OpenSwitch.
Examples	<p>Shuts down the RCM named "osw_primary_rcm."</p> <pre>1> rp_rcm_shutdown osw_primary_rcm 2> go</pre> <p>Returns:</p> <pre>gap: DEBUG: spid 10: coord_rcm_notif succeeded. Msg 20108, Level 16, State 0: Server 'gap': rp_rcm_shutdown: The Primary RCM osw_primary_rcm will be shutdown. (return status = 0)</pre>
Usage	<p>Shuts down an RCM through OpenSwitch. The RCM does not have to have been started by OpenSwitch to be shut down using <code>rp_rcm_shutdown</code>.</p> <p>After issuing <code>rp_rcm_shutdown</code>, execute <code>rp_rcm_list</code> and <code>ps</code> to verify that the RCM has shut down.</p>
See also	<code>rp_rcm_list</code> , <code>rp_rcm_startup</code>

rp_rcm_startup

Description	Starts an RCM using the specified path and configuration file.
Syntax	<code>rp_rcm_startup [rcm_path, rcm_cfg, rcm_log, rcm_retries, rcm_redundant]</code>
Parameters	<p><i>rcm_path</i> Path to the RCM binary.</p> <ul style="list-style-type: none">• If not provided (NULL), the RCM binary is executed using the value of the <i>RCM_PATH</i> parameter set in the OpenSwitch configuration file. If <i>RCM_PATH</i> is not set, the RCM binary is executed from <i>\$OPENSWITCH/bin/rcm</i> on UNIX and from <i>%OPENSWITCH%\bin\rcm</i> on Windows.• When you provide <i>rcm_path</i>, the RCM executable is invoked from the specified location. You must have execution permission on the RCM binary. <p><i>rcm_cfg</i> Absolute path to the RCM configuration file. If not provided, the value specified by the <i>RCM_CFG_FILE</i> parameter in the OpenSwitch configuration file is used. You must have read access on this file.</p> <p><i>rcm_log</i> Absolute path to the RCM log. If not provided, the value specified by the <i>RCM_LOG_FILE</i> parameter in the OpenSwitch configuration file is used.</p> <p><i>rcm_retries</i> An integer value to specify the number of times OpenSwitch restarts the RCM if it fails. If NULL, the value specified by <i>RCM_RETRIES</i> parameter in the OpenSwitch configuration file is used.</p> <p><i>rcm_redundant</i> An Boolean value (1 is “true” and zero (0) is “false”) to specify whether or not the RCM being started is a redundant RCM. If 1, the RCM binary is executed with the “-R” option. If zero (0), the RCM binary is not executed with the “-R” option.</p>
Examples	<p>Starts the RCM with the default RCM binary, trying twice, with no redundancy.</p> <pre>1> rp_rcm_startup NULL, /scratch/15.0_bld2/OpenSwitch-15_0/config/rcm.cfg, /scratch/15.0_bld2/OpenSwitch-15_0/logs/rcm.log, 2, 0 2> go</pre> <p>Returns:</p> <pre>Msg 20107, Level 16, State 0: Server 'monsoon_OSW':rp_rcm_startup:</pre>

```
The RCM has been started.  
(return status = 0)
```

Usage

- *RCM_AUTOSTART* must be set to 1 in the OpenSwitch configuration file to run this command. Use *rp_set* to dynamically configure *RCM_AUTOSTART*.
- Verify that *RCM_CFG_FILE* and *RCM_LOG_FILE* are pointing to valid locations and that the files to which these parameters point have the proper access permissions before executing *rp_rcm_startup*.
- Always check the OpenSwitch error log after you run *rp_rcm_startup* to verify that the RCM has started; the command can sometimes return a success before the RCM has completely started. When *rp_rcm_startup* is successful, you see this message in the OpenSwitch error log:

```
INFO: spid 19: rp_rcm_startup: The RCM has been started.  
INFO: spid 22: rcm__thread: Service thread spawned.
```

- After you start the RCM, use *rp_rcm_list* to verify that the RCM has started.

See also

rp_rcm_shutdown, *rp_rcm_list*

rp_replay

Description Replays SQL statements after the connection is established to the failover server and before a new client request is accepted using that connection. Use *rp_replay* to specify to the connection before the failover occurs the SQL statements to be replayed.

Syntax *rp_replay spid, action, name, sql, canfail*

Parameters *spid*

Either an OpenSwitch process ID, or NULL.

action

The action to be performed on the SQL statements. Action can be ADD, DELETE, CLEAR, or SHOW. If the action is CLEAR, the *name* parameter must be NULL; if the action is SHOW, the *name* parameter can be NULL. See the Usage section for more information.

name

The name of the SQL statement, which must be unique within a thread

sql

A SQL statement. The maximum length of the SQL statement is 2048 bytes. Supply this parameter when the action is ADD.

canfail

Checks whether or not the statement is allowed to fail. If you do not supply this parameter, *canfail* has a value of zero (0). If *canfail* is 1, the connection is still considered properly failed over even if the statement did not execute on the remote server. Supply this parameter only when the action is ADD.

Examples

Example 1 Creates multiple *rp_replay* SQL statements for *spid* 8.

```
1> rp_replay 8, "add", "a1", "create table temp1(a int, b char)", 0
2> go
(return status = 0)

1> rp_replay 8, "add", "a2", "insert into temp1 values (15, 'w')", 1
2> go
(return status = 0)

1> rp_replay 8, "add", "a3", "insert into temp1 values (17, 'x')", 0
2> go
(return status = 0)

1> rp_replay 8, "add", "a4", "insert into temp1 values (22, 'y')", 0
2> go
(return status = 0)
```

```

1> rp_replay 8, "add", "a5", "insert into temp1 values (78, 'z')", 0
2> go
(return status = 0)

```

Example 2 Displays the `rp_replay` SQL statement for “a2” and *spid* 8.

```

1> rp_replay 8, "show", "a2"
2> go

```

Returns:

spid	SQL_Name	SQL_Statement	canfail
8	a2	insert into temp1 values (15, 'w')	1

```

(return status = 0)

```

Example 3 Displays the `rp_replay` SQL statements for *spid* 8.

```

1> rp_replay 8, "show", NULL
2> go

```

Returns:

spid	SQL_Name	SQL_Statement	canfail
8	a1	create table temp1(a int, b char)	0
8	a2	insert into temp1 values (15, 'w')	1
8	a3	insert into temp1 values (17, 'x')	0
8	a4	insert into temp1 values (22, 'y')	0
8	a5	insert into temp1 values (78, 'z')	0

```

(return status = 0)

```

Example 4 Displays all `rp_replay` SQL statements for “a1.”

```

1> rp_replay NULL, "show", "a1"
2> go

```

Results:

spid	SQL_Name	SQL_Statement	canfail
8	a1	create table temp1(a int, b char)	0
11	a1	insert into temp1 values (97, 'q')	0

```

(return status = 0)

```

Example 5 Adds the same `re_replay` SQL statement to all *spids* that are currently active in the OpenSwitch server.

```

1> rp_replay NULL, "add", "a7", "set textsize 17889", 1
2> go
(return status = 0)

```

```
1> rp_replay NULL, "show", "a7"
2> go
```

Returns:

spid	SQL_Name	SQL_Statement	canfail
8	a7	set textsize 17889	1
11	a7	set textsize 17889	1

(return status = 0)

Example 6 Displays all SQL statements for all *spids*.

```
1> rp_replay NULL, "show", NULL
2> go
```

Returns:

spid	SQL_Name	SQL_Statement	canfail
8	a1	create table temp1(a int, b char)	0
8	a2	insert into temp1 values (15, 'w')	1
8	a3	insert into temp1 values (17, 'x')	0
8	a4	insert into temp1 values (22, 'y')	0
8	a5	insert into temp1 values (78, 'z')	0
8	a7	set textsize 17889	1
11	a1	insert into temp1 values (97, 'q')	0
11	a2	insert into temp1 values (98, 'r')	1
11	a8	insert into temp1 values (100, 's')	0
11	a9	delete from temp1 where a = 22	0
11	a10	set textsize 32567	0
11	a3	set rowcount 2	1
11	a7	set textsize 17889	1

(return status = 0)

Usage

- `rp_replay` replays SQL statements after the connection is established to the failover server and before a new client request is accepted using that connection.
- `rp_replay` parameters allows users to add, delete, clear, and display the SQL statements.
- If the user gives NULL for the *spid* parameter, the action is applied only to existing *spids* and not new *spids*.
- DELETE removes the SQL statement based on the *name* parameter, while CLEAR removes all SQL statements attached to the specified *spid*.

Messages

- The user has supplied an *action* parameter other than ADD, CLEAR, DELETE, or SHOW:

```
rp_replay: The @action parameter must be ADD, CLEAR,
DELETE, or SHOW surrounded by double quotes.
```
- The user has not supplied the SQL statement to add:

```
rp_replay: The @sql parameter must be supplied with
an "add" action
```
- The user has supplied the *sql* parameter when the *action* parameter is not ADD:

```
rp_replay: The @sql parameter must be supplied with
an "add" action.
```
- The user has supplied the *canfail* parameter when the *action* parameter is not ADD:

```
rp_replay: The @canfail parameter must be supplied
with an "add" action
```
- The *action* is CLEAR but the *name* parameter is not NULL:

```
rp_replay: @name should be NULL for CLEAR action. To
remove a specific statement, use action=delete
instead.
```
- The user is trying to delete a SQL statement that does not exist:

```
thrd_sql_rem: Attempt to remove nonexistent SQL `a1'

Msg 20075, Level 13, State 0:

Server 'test_osw':
rp_replay:
Internal thrd_sql_rem error, see log file
(return status = -1)
```
- The user is trying to add a SQL statement with a nonunique name:

```
Message - test_osw: INFO: spid 10: thrd_sql_add:
@name should be unique, 'insert into templ values
(15, 'w')' already have name 'a2' for spid '8'.
Unable to add statement for spid '8'.

Msg 20075, Level 13, State 0:
Server 'test_osw':
rp_replay:
```

```
Internal thrd_sql_add error, see log file  
(return status = -1)
```

rp_rmon

Description Displays the current set of attribute/value pairs being used by the resource governor thread.

Syntax `rp_rmon`

Parameters None.

Examples `rp_rmon`

Returns:

busy	attribute	entry	action
-----	-----	-----	-----
300	hostname	clientsrv1.sample.com	CANCEL
300	hostname	clientsrv2.sample.com	KILL
30	username	batch[0-9]%	CANCEL
30	appname	isql	CANCEL

- Usage**
- Displays the current set of attribute/value pairs being used by the resource governor thread.
 - To change the entries shown by `rp_rmon` while the server is running, you must edit the configuration file and restart OpenSwitch.

See also `rp_cfg`

rp_server_help

Description Displays the name, status, HA-type, configuration storage definition, CMON username and CMON password for the specified server as defined in the [SERVER] section.

Syntax rp_server_help [server]

Parameters server
Name of the OpenSwitch server for which to display properties. If you do not provide this parameter, OpenSwitch displays the properties of all OpenSwitch servers in the [SERVER] section of the OpenSwitch configuration file.

Examples **Example 1** Displays the properties of server “itan1.”

```
1> rp_server_help itan1
2> go
```

Returns:

server	status	type	cfg_storage	cmon_user	cmon_pwd
itan1	UP	NULL	1	NULL	NULL

(1 row affected)
(return status = 0)

Example 2 Displays the properties of all the servers known to the OpenSwitch on which rp_server_help is executed.

```
1> rp_server_help
2> go
```

Returns:

server	status	type	cfg_storage	cmon_user	cmon_pwd
DEFAULT	UP	NULL	0	NULL	NULL
itan1	UP	NULL	1	NULL	NULL
callie1	UP	NULL	1	NULL	NULL

(3 rows affected)
(return status = 0)

Usage Used to find out the properties of a specific server, or of all the servers known to a specific OpenSwitch.

rp_server_status

Description	Sets or displays the status of any remote server that is defined in the [SERVER] section of the OpenSwitch configuration file.
Syntax	rp_server_status [server, status]
Parameters	<p><i>server</i></p> <p>The name of a remote server as listed in the <i>interfaces</i> file on UNIX, or <i>sql.ini</i> on Windows of OpenSwitch. The server must also be defined in the [SERVER] section of the OpenSwitch configuration file. If a server name is supplied, but the <i>status</i> is not supplied, the status of the specified server is displayed.</p> <p><i>status</i></p> <p>The disposition to which the server is to be changed. If <i>status</i> is supplied, but a server is not, then the current status of all servers is changed.</p>

Table 6-9: Status values for rp_server_status

Status	Description
PRE_UP	<p>Mutually-aware-specific server status. The server is either in the process of being marked as UP, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to UP on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a server's status to PRE_UP.</p> <hr/>
UP	The server is immediately available for use.
PRE_DOWN	<p>Mutually-aware specific server status. The server is either in the process of being marked as DOWN, or has encountered a problem during that process. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to DOWN on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a server's status to PRE_DOWN.</p> <hr/>
DOWN	The server is unavailable, and is not considered for use by any new client connections established to OpenSwitch.

Status	Description
PRE_LOCKED	<p>Mutually-aware specific server status. The server is either in the process of being marked as LOCKED, or has encountered a problem during that process. doing so. Check the error log to troubleshoot the problem. After you resolve the problem, manually set the server status to LOCKED on one of the mutually-aware companion OpenSwitch servers. The command will be propagated to the other OpenSwitch companion if it is running.</p> <hr/> <p>Warning! Do not manually set a server's status to PRE_LOCKED.</p> <hr/>
LOCKED	The server is available, but any new incoming connections actively being connected through the pool are blocked (or stopped) until the status is changed to UP or DOWN. Blocked connections appear to the client applications to have stopped responding until the pool is unlocked.

Examples

Example 1 Displays the current status of all servers:

```
1> rp_server_status
2> go
```

Returns:

```
server          status
-----
DEFAULT         UP
SYB_SERV1       UP
SYB_SERV2       UP
SYB_SERV3       UP
```

Example 2 Displays the current status of server SYB_SERV1:

```
1> rp_server_status "SYB_SERV1"
2> go
```

Returns:

```
server          status
-----
SYB_SERV1       UP
```

Example 3 Sets the status of SYB_SERV1 to LOCKED:

```
1> rp_server_status "SYB_SERV1", "LOCKED"
2> go
```

Returns:

```
server          status
-----
SYB_SERV1       LOCKED
```

Usage

- Changing the status of a server has no effect on existing connections.
- The special server, `DEFAULT`, applies to all servers that have not had their status explicitly set (either through the configuration file or a call to `rp_server_status`). That is, if a server that is not currently listed in `rp_server_status` is added to an existing pool, that server inherits the status of the `DEFAULT` server.
- Connections that are currently blocked on a `LOCKED` pool continue to remain blocked until either the pool is unlocked or the client application disconnects. This means that any administrative requests made of the connection, such as a call to `rp_switch`, or `rp_stop`, are queued until the pool changes status.
- The server status set for a pool using `rp_pool_server_status` or `rp_pool_addserver` takes precedence over a server's status set within that pool using `rp_server_status`.
- If a server does not have a pool-specific status, it inherits the generic status set using `rp_server_status`.
- If you issue `rp_pool_status` to set the pool status on a mutually-aware OpenSwitch server, the command is propagated to the companion OpenSwitch if it is running. The new pool status is also recorded in the mutually-aware configuration tables on the Adaptive Servers. If these steps fail, the pool status is reset to its original value and an error message is logged.
- If `FREEZE_CFG_ON_FAIL` is enabled and the network fails between the local OpenSwitch server and the companion OpenSwitch server, you can use `rp_pool_status` only to display, not set, a pool's status. This prevents the companion OpenSwitch servers from switching client connections to different Adaptive Servers while the network between OpenSwitch servers is not responding, which could cause data loss when replication is performed in only one direction. `rp_pool_status` allows you to reset the pool status when the network is restored between the companion OpenSwitch servers, or if `FREEZE_CFG_ON_FAIL` is disabled.

Messages

- The server does not exist within OpenSwitch:

```
rp_server_status: There is no such server 'server'.
```

To display a list of existing pools, use `rp_server_status` with a server parameter of `NULL`.

- This mutually-aware OpenSwitch server has detected a network problem with its companion, and prohibits all future status changes until the network is restored and the status of the companion can be verified:

```
rp_server_status: Status cannot be set/changed until  
connectivity is restored with the companion  
OpenSwitch site or the FREEZE_CFG_ON_FAIL parameter  
is turned OFF.
```

See also

`rp_pool_addserver`, `rp_pool_server_status`

rp_set

Description	Sets or displays configuration parameters.
Syntax	<code>rp_set [parm_name, parm_value]</code>
Parameters	<p><i>parm_name</i> Name of a parameter as listed in the configuration file.</p> <p><i>parm_value</i> Value to which the parameter is to be set. The meaning of the value depends on which parameter is being set.</p>

Examples **Example 1** Displays the value of the TEXTSIZE configuration parameter.

```
rp_set TEXTSIZE
```

Example 2 Sets the TEXTSIZE configuration parameter to 1MB.

```
rp_set TEXTSIZE, 1048576
```

Example 3 Displays all the configuration options and their current values. Only the first 27 characters of the configuration parameter and the first 50 characters of the value display.

```
1> rp_set
2> go
```

Returns:

parameter	value
-----	-----
SERVER_NAME	posw
MUTUAL_AWARE	1
MUTUAL_CLUSTER	owsCluster
PRIMARY_COMPANION	1
FREEZE_CFG_ON_FAIL	0
LOG_FILE	posw.log
LOT_TO_OS	0
CFG_FILE	./posw.cfg
CUSTOM_SCRIPT	/usr/u/johndoe/custom.sh
MANUAL_SCRIPT	/usr/u/johndoe/manual.sh
RCM_CFG_FILE	/oswitch/config/
RCM_PATH	NULL
RCM_AUTOSTART	0R
CM_SECONDARY	0R
CM_RETRIES	0R
CM_LOG_FILE	/oswitch/bin/rcm
UPDATE_CFG	1
DEBUG_FILE	posw.dbg

ADMIN_USER	sa
ADMIN_PASSWORD	*****
INTERFACES	NULL
CHARSET	iso_1
CONNECTIONS	1000
CON_TRACE	0
CTX_TRACE	0
SRV_TRACE	0
TRUNCATE_LOG	1
SITE_PASSTHRU	1
ECHO_LOG	1
DEBUG	eCf
FULL_PASSTHRU	0
RMON	0
RMON_INTERVAL	10
SEC_PRINCIPAL	NULL
SHOW_SPID	0
STACKSIZE	40960
TCP_KEEPAIVE	1
TCP_NODELAY	1
COORD_MODE	AVAIL
COORD_USER	switch_coord
COORD_PASSWORD	*****
TEXTSIZE	102400
SUPPRESS_CHARSET	1
SUPPRESS_DBCTX	1
SUPPRESS_LANG	1
CURSOR_PREREAD	20
OPTIMIZE_TEXT	1
MAX_LOGSIZE	4194304
MSGQ_SIZE	2048
API_CHECK	1
HAFAILOVER	0
SQL_WRAP	80
CACHE_THREADS	100
BCP_LOGGED	0
MAX_PACKETSIZE	2048
COORD_TIMEOUT	30
LOGIN_TIMEOUT	60
RESPONSE_TIMEOUT	3600
CMON	1
CMON_USER	sa
CMON_PASSWORD	*****
CMON_WAITFOR_DELAY	3600
SWITCH_AT_LOGIN_TIMEOUT	0
MAX_LOG_MSG_SIZE	1024

```

USERNAME_PASSWORD_ENCRYPT      0
USE_AND_TO_POOL_ATTRIB        0
USE_AND_TO_RMON_ATTRIB        0
USE_DONEINPROCS                0
SHOW_CONNECT_ERROR             0
NOWAIT_ON_LOCKED               0
SVR_FAIL_ACTION                 DEFAULT
NET_FAIL_ACTION                 DEFAULT
CMP_FAIL_ACTION                 DEFAULT
CMON_FAIL_ACTION                DEFAULT
PING_THREAD                     1
PING_BINARY                     /usr/sbin/ping
PING_WAIT                       10
PING_RETRIES                    1
RPC_SETFMT                      0
TEST_PING_COUNT                 5
(84 rows affected)
(return status = 0)

```

Usage

- Use `rp_set` to query or set the value of a configuration parameter.
- Not all configuration parameters are dynamic. For example, parameters such as maximum number of connection (CONNECTIONS) and whether or not the resource monitor thread is running (RMON) are read by OpenSwitch only at start-up.

Messages

- The parameter name supplied does not exist or the value supplied is illegal for the parameter name:

```
rp_set: Invalid parameter name or value.
```

See the OpenSwitch error log for more information.

- The parameter specified is a static parameter that cannot be changed using `rp_set`:

```
rp_set: Cannot modify Static or Non-Existing
parameter <parameter> at runtime. Please refer to
OpenSwitch Documentation for additional information.
```

To enable the parameter, restart OpenSwitch with the new value in the configuration file.

See also

The [CONFIG] section in Chapter 4, “Using the Configuration File.”

rp_set_srv

Description	Responds to <i>spid</i> waiting for coordination module response.
Syntax	<code>rp_set_srv <i>spid</i>, <i>srv_name</i></code>
Parameters	<p><i>spid</i></p> <p>The OpenSwitch process identifier of the connection waiting for a response from a coordination module.</p> <p><i>srv_name</i></p> <p>The name of the remote server to which the connection, represented by <i>spid</i>, should be switched. Passing a server name of NULL or "" causes the <i>spid</i> to switch to the next available server according to the mode of the pool to which it belongs.</p>
Examples	<p>Example 1 Connects <i>spid</i> 8 to SYB_SERV3.</p> <pre>rp_set_srv 8, "SYB_SERV3"</pre> <p>Example 2 Connects <i>spid</i> 8 to the next available server in its pool.</p> <pre>rp_set_srv 8, ""</pre>
Usage	<ul style="list-style-type: none">• When a connection requests a server name from a coordination module, for any reason (including a login attempt or a failure detected from an existing server), the failing connection issues an <code>np_req_srv</code> notification procedure call that is detected by the coordination module (see “<code>np_req_srv</code>” on page 206). The requesting connection suspends until the coordination module responds with a call to <code>rp_set_srv</code>, <code>rp_switch</code>, or <code>rp_kill</code>.• <code>rp_set_srv</code> is intended for use by the coordination module to set the server for a <i>spid</i> through <code>cm_set_srv</code>, but an administrator can also manually set the server for a <i>spid</i> if the <i>spid</i> is at a “SERVER REQ” state. This can happen if <code>COORD_MODE</code> is set to ALWAYS and no coordination module is running.• You can list the set of <i>spids</i> awaiting a response from the coordination module using the Open Server built-in registered procedure <code>sp_ps</code>. These <i>spids</i> display a <i>sleep_label</i> of “SERVER REQ”.• Calling <code>rp_req_srv</code> on a <i>spid</i> that is not actively waiting for a response from a coordination module has no effect.
See also	<code>rp_kill</code> , <code>rp_switch</code>

rp_showquery

Description	Displays query being executed by a <i>spid</i> . <i>rp_showquery</i> works only when OpenSwitch is not in FULL_PASSTHRU mode.
Syntax	<i>rp_showquery spid</i>
Parameters	<i>spid</i> The OpenSwitch process identifier of the connection executing a query.
Examples	<pre>1> rp_showquery 8 2> go</pre> <p>Returns:</p> <pre>Statement ----- sp_who</pre>
Usage	<ul style="list-style-type: none">• Use <i>rp_who</i> to list valid <i>spids</i>.• Output is displayed only for <i>spids</i> that are actively processing a query (the state field of <i>rp_who</i> shows BUSY), or executing registered procedures or SQL queries. Cursors are not currently supported. <p>Messages</p> <ul style="list-style-type: none">• <i>spid x</i> is not a valid <i>spid</i> as listed in <i>rp_who</i>: <pre>rp_showquery: spid #x is invalid.</pre>
See also	<i>rp_who</i>

rp_shutdown

Description	Shuts down the OpenSwitch server.
Syntax	<code>rp_shutdown</code>
Usage	Use with caution. No verification or authentication is performed before the server is shut down.
See also	<code>rp_stop</code> , <code>rp_switch</code>

rp_start

Description	Starts a group of previously stopped connections.
Syntax	<code>rp_start pool_name, server, spid</code>
Parameters	<p><i>pool_name</i> Name of the pool in which to restart connections. If you do not supply a parameter, or use NULL, all pools are started.</p> <p><i>server</i> Indicates that all connections using the remote server are to be started. If you do not supply a server name, or use NULL, all servers are started.</p> <p><i>spid</i> The OpenSwitch process ID of the connection to be started. If you do not supply an argument, or use NULL, all connections are started.</p>
Examples	<p>Example 1 Starts previously stopped <i>spid</i> number 8.</p> <pre>rp_start NULL, NULL, 8</pre> <p>Example 2 Starts all connections that are currently using remote server SYB_SERV1.</p> <pre>rp_start NULL, "SYB_SERV1", NULL</pre> <p>Example 3 Starts all connections established through POOL_A.</p> <pre>rp_start "POOL_A", NULL, NULL</pre> <p>Example 4 Starts all connections established to SYB_SERV1 through POOL_A.</p> <pre>rp_start "POOL_A", "SYB_SERV1", NULL</pre>
Usage	Attempting to start a connection that was not previously stopped has no effect.
Messages	<p>Messages</p> <ul style="list-style-type: none"> The <i>pool_name</i> does not exist. Use <code>rp_pool_help</code> to list valid pool names: <pre>rp_start: Invalid pool name 'pool_name'.</pre> The <i>server</i> does not exist: <pre>rp_start: Invalid server name 'server'.</pre> <p>Use <code>rp_server_status</code> to list valid servers.</p>
See also	<code>rp_pool_help</code> , <code>rp_server_status</code> , <code>rp_stop</code> , <code>rp_who</code>

rp_stop

Description	Stops a group of connections from issuing new queries.
Syntax	rp_stop [pool_name], [server], [spid], [ign_tran], [ign_fail]
Parameters	<p><i>pool_name</i></p> <p>The name of the pool in which connections are to be stopped. If you do not supply a parameter, or use NULL, all pools are stopped.</p> <p><i>server</i></p> <p>Indicates that all connections using the remote server are to be stopped. If you do not supply a server name or use NULL, all servers are stopped.</p> <p><i>spid</i></p> <p>The OpenSwitch process ID of the connection to be stopped. If you do not supply an argument, or use NULL, all connections are stopped.</p> <p><i>ign_tran</i></p> <p>Indicates whether or not rp_stop ignores transaction state when pausing a connection.</p>

Table 6-10: Values for ign_tran

Value	Description
1	All connections are stopped whether or not they are involved in an open transaction. This is the default value if ign_tran is not supplied.
0	All connections are stopped as soon as they complete their current transaction.

<i>ign_fail</i>	Indicates whether or not stopped connections ignore remote server failures while they are stopped.
-----------------	--

Table 6-11: Values for ign_fail

Value	Description
1	If an Adaptive Server actively being used by a stopped connection fails, an attempt is made to reestablish the connection silently without notifying a coordination module when rp_start is issued.
0	If an Adaptive Server actively being used by a stopped connection fails, the normal failover process proceeds for the connection as soon as rp_start is issued. This is the default value.

Examples	Example 1 Stops <i>spid</i> number 8 regardless of the pool or server it is associated with.
----------	---

```
rp_stop NULL, NULL, 8
```

Example 2 Stops all connections currently using remote server SYB_SERV1.

```
rp_stop NULL, "SYB_SERV1", NULL
```

Example 3 Stops all connections established through POOL_A.

```
rp_stop "POOL_A", NULL, NULL
```

Example 4 Stops all connections established to server SYB_SERV1 through pool POOL_A.

```
rp_stop "POOL_A", "SYB_SERV1", NULL
```

Example 5 Stops *spid* number nine in POOL_A when it is finished with its current transaction.

```
rp_stop "POOL_A", NULL, 9, 0
```

Usage

- `rp_stop` broadcasts a request to all connections matching the *pool_name*, *server*, and *spid* parameters. Each connection polls to see if it has received a stop request immediately before and immediately after processing a new client query, at which time the connection sleeps (or appears to the client to have stopped responding) until an `rp_start` request is issued.
- Use the Open Server registered procedure `sp_ps` to determine if connections that are not responding are indicated by a Sleep Label of “COORD_STOP.” See the Sybase Open Server documentation for more information about using `sp_ps`.

Note In the return data for `sp_ps`, the Sleep Label is a column that describes the sleep event. `COORD_STOP` is a status that OpenSwitch sets in the thread when an administrator or CM stops the threads using `cm_stop` or `rp_stop`. Use `rp_start` or `cm_start` to “wake up” threads that have been put to sleep.

However, since connections only actually respond to an `rp_stop` request when starting or completing communication with the remote server (for example, a query is issued), `COORD_STOP` does not display for connections that have remained idle since the request was issued.

- `rp_stop` applies only to connections that are already established to OpenSwitch.
- Attempting to stop a connection that is already stopped has no effect.

Messages

- The supplied pool name does not exist in OpenSwitch:

```
rp_stop: Invalid pool name 'pool_name'.
```

Use `rp_pool_help` to list valid pool names.

- The supplied server does not exist in OpenSwitch:
`rp_stop: Invalid server name 'server'.`

Use `rp_server_status` to list valid servers.

rp_switch

Description	Manually switches user connections to an alternate Adaptive Server.
Syntax	<code>rp_switch [pool_name], [src_server], [spid], [dst_server], [grace_period], [force]</code>
Parameters	<p><i>pool_name</i></p> <p>All connections established through the <i>pool_name</i> you specify are switched to the server specified by <i>dst_server</i>. If you do not supply this parameter, or specify NULL, all pools are assumed.</p> <p><i>src_server</i></p> <p>All the connections that are currently established to this remote server (<i>src_server</i>) switch to the <i>dst_server</i>. If you do not supply this parameter, or specify NULL, all servers are assumed.</p> <p><i>spid</i></p> <p>The OpenSwitch <i>spid</i> to be switched to the remote server <i>dst_server</i>. If you do not supply this parameter, or specify NULL, all <i>spids</i> are assumed.</p> <p><i>dst_server</i></p> <p>The remote server to which all connections identified by <i>pool_name</i>, <i>src_server</i>, and <i>spid</i> should be switched. If you do not supply this parameter, or specify NULL or "", the connections are switched to the first available server as identified by their associated pool.</p> <p>Use care when specifying this parameter. No verification is performed for <i>dst_server</i>. Passing an invalid value causes all incoming client connections to be lost.</p> <p><i>grace_period</i></p> <p>The maximum number of seconds to wait before forcefully switching busy connections. A value of 0 indicates that no grace period is to be enforced.</p> <p><i>force</i></p> <p>If passed with a value of 1, all connections are forcefully switched, even if they are currently busy (either actively in the middle of communicating with a remote server, or in the middle of an open transaction). If you do not supply a value, or specify NULL, the value defaults to 0.</p>
Examples	<p>Example 1 Causes OpenSwitch <i>spid</i> number 8 to be switched to the remote server SYB_SERV1. If the <i>spid</i> does not successfully switch in 60 seconds, its current query is canceled and the client receives a “deadlock” message, and the connection is then switched.</p>

```
rp_switch NULL, NULL, 8, "SYB_SERV1", 60, 0
```

Example 2 Switches all connections established through POOL_A to SYB_SERV1 immediately. Any busy connections receive a “deadlock” message and the current query is canceled.

```
rp_switch "POOL_A", NULL, NULL, "SYB_SERV1", 0, 1
```

Example 3 Switches all connections to SYB_SERV1, established through POOL_A to SYB_SERV2, within 60 seconds.

```
rp_switch "POOL_A", "SYB_SERV1", NULL, "SYB_SERV2", 60,  
0
```

Example 4 Switches all connections in POOL_A to the next available server (as defined by the mode of the pool) within 60 seconds.

```
rp_switch "POOL_A", NULL, NULL, NULL, 60, 0
```

Example 5 Switches all existing connections to OpenSwitch to the next available server within the pool associated with each connection. There is no limit on how long the switching process is to take.

```
rp_switch
```

Usage

- A call to `rp_switch` causes a switch request to be issued to all connections matching *pool_name*, *src_server*, or *spid*. The switch request is processed by each connection under the following conditions:
 - If the connection is completely idle, it is switched immediately.
 - If the connection is busy (either communicating with a remote server or involved in an open transaction) and *grace_period* is zero (0) and force is zero (0), the connection switches as soon as it becomes idle.
 - If the connection is busy, and *grace_period* is a positive value and force is 0, the connection switches as soon as it becomes idle, or if the number of seconds specified in *grace_period* pass before it becomes idle, the current query is canceled, and a “deadlock” message is issued to the client, and then its connection is switched.
 - If the connection is busy and force has a value of 1, the connection immediately cancels its query, receives a “deadlock” message, then switches the connection.
- Use caution when specifying the *dst_server* parameter. No verification is performed for this parameter. The administrator must verify that the *dst_server* is UP and that its entry exists in the *sql.ini* (Windows) or *interfaces* (UNIX) file before executing `rp_switch` to switch connections to it.

If the *dst_server* being passed to does not exist, is not running, or cannot be connected to, all the switched connections, as well as the incoming client connections are lost.

- *dst_server* does not need to be a server within the pool of a given connection, or even a server within any pool. It simply must be a valid server.
- If *force* is 1, then *grace_period* must be 0, since *grace_period* does not make sense in this context.
- A switch request issued to a connection that is blocked due to either a call to *rp_stop*, a LOCKED pool, or a LOCKED *src_server* is processed as soon as the connection becomes unblocked. Existing connections that are not blocked are switched immediately to the next available server, or the *dst_server* if it is specified.
- The user performing the switch cannot do so while passing through *OpenSwitch*. The switch does not complete because the connection is not idle.

Messages

- An invalid value was supplied for the *force* parameter:

```
rp_switch: @force must be 0 or 1
```

Valid values are 0 and 1.

- A nonzero value was supplied for *grace_period*, and a value of 1 was supplied for the *force* parameter:

```
rp_switch: @grace_period must be 0 when @force is 1.
```

- The supplied *pool_name* does not exist:

```
rp_switch: Invalid pool name 'pool_name'.
```

Use *rp_pool_help* to list valid pools.

- The supplied server name is not known among the existing remote servers:

```
rp_switch: Invalid source server name 'src_server'.
```

Use *rp_server_status* to list valid servers.

- The procedure ran normally, and *n* spids have been requested to switch:

```
rp_switch: Queued switch request for n spids.
```

See also

rp_who

rp_traceflag

Description Enables or disables SRV_TRACE flags for debugging messages.

Syntax `rp_traceflag [options, {on|off}]`

Parameters

options

A list of one or more single-character option flags.

The following table shows the valid debugging options. These options are identical to the options you can use with the -s flag at the command line.

Value	Displays
a	TDS attention packets
d	TDS data information
e	Server events
h	TDS header information
m	Message queue usage
n	Network driver information and TCL requests
p	Network driver parameter information
q	Run queue information
r	Network driver data information
s	Network driver memory information
t	TDS tokens
w	TCL wake-up request

on

Turns on debugging options, which causes debugging messages to be dumped to the error log file.

off

Turns off debugging options.

Examples

Example 1 Displays all debugging flags and their current state.

`rp_traceflag`

Returns:

flag	description	state

a	TDS attention packets	on
d	TDS data information	on
e	Server events	on
h	TDS header information	on
m	Message queue usage	on
n	Network driver information and TCL	

	requests	on
p	Network driver param information	on
q	Run queue information	on
r	Network driver data information	on
s	Network driver memory information	on
t	TDS tokens	on
w	TCL wakeup request	on

Example 2 Switches off TDS data information and server events flags.

```
rp_traceflag "de", off
```

Returns:

flag	description	state

d	TDS data information	off
e	Server events	off

Example 3 Displays the current status of the -d flag.

```
rp_traceflag d
```

Returns:

flag	description	state

d	TDS data information	off

Usage

Messages

- If the *state* parameter is supplied, then the *flags* parameter cannot be NULL:

```
rp_traceflag: @flags cannot be NULL if @state is
supplied
```

- The valid value for the *state* parameter is either on or off:

```
rp_traceflag: @state must be on or off
```

rp_version

Description Returns the OpenSwitch version number.

Syntax `rp_version`

Examples

```
1> rp_version
2> GO
```

Returns:

```
Version
-----
Sybase OpenSwitch/15.0/P/SPARC/Solaris 2.8/0/OPT/
Fri Sep  9 15:55:28 2005
(1 row affected)
```

Usage `rp_version`

rp_who

Description Displays detailed information about user connections.

Syntax `rp_who [spid]`

Parameters *spid*
The OpenSwitch *spid* value to be displayed.

Examples

spid	state	actions	user	host	database	current	next	pool
1	BUSY	NULL	elyse	NULL	testdb	SYB_SRV1	SYB_SRV1	POOL1
2	BUSY,TRAN	NULL	athena	NULL	master	SYB_SRV1	SYB_SRV1	POOL

Usage Table 6-12 describes the contents of each column returned by `rp_who`.

Table 6-12: Columns returned by *rp_who*

Column	Description
spid	The internal Open Server process ID of the user connection.
rspid	The spid of the user connection on the remote server. This display only if SHOW_SPID is set to 1 in your OpenSwitch configuration. Otherwise, it displays as NULL.
state	A comma-delimited list of connection state information. Table 6-13 on page 201 describes the possible values for state.
actions	Pending actions that have been queued for the thread. These actions are typically initiated by an administrator by calling such registered procedures as <code>rp_switch</code> , but some of them may arise internally due to things like an Adaptive Server failing. Table 6-14 on page 202 describes the possible values for actions. Zero or more of these may be active at any time.
application	The name of the application used to establish the client connection.
user	The user name of the client connection.
host	The host machine from which the client connection was established.
database	The current database context of the user connection.
current	The name of the remote server actively being used by the client connection.
next	The name of the remote server that the client connection should be using. This information may differ from that in <code>current</code> if a switch is actively being performed.
pool	The name of the pool the user connection is using.
function	A debugging field. This describes which ctlib function this connection may be blocked on.

Table 6-13 describes the state of each connection returned by `rp_who`.

Table 6-13: Values for state

Value	Description
ATTNWAIT	Internal flag.

Value	Description
BLOCKED	The client thread is currently performing a blocking activity, such as a read or a write, upon its outgoing connection.
BUSY	The client is actively communicating with a remote server, either sending or receiving results. While the connection is in this state, it is considered busy and is not be switched by a nonforced administrative switch request (see <code>rp_switch</code> on page 195).
CACHE	The thread owning the connection is a CACHE thread and does not belong to a “real” client.
CANCAN	Internal flag.
CANCELED	Indicates that the GOTATTN flag has been acted upon and the current query has been canceled.
CLOSED	Internal state indicating that the client connection is currently closed.
GOTATTN	Flag that is set when the client has raised an attention via a cancel request (such as a call to <code>dbcancel()</code> or <code>ct_cancel()</code>). This state should be followed later by a CANCELED state.
IGNOREMSG	Internal state. This state indicates that the connection is ignoring error or informational messages received from the remote server. It is set while a connection is being switched to another server to ensure that the client does not see the database context change messages issued while logging in to the remote server.
KILL	Indicates that the connection is in the process of being killed and will soon be removed from OpenSwitch.
LOGIN	Set while the connection is actively logging in to a remote server.
LOGINFAIL	Used to indicate that the connection was denied to the remote server due to things like an invalid password. This is a transient state that is usually immediately followed by the client connection being disconnected from OpenSwitch.
LOST	Indicates that the client’s outgoing connection to a remote server has been lost. This state is transitory and remains in effect until OpenSwitch either fails the connection over or disconnects the client.
STOPPED	The client is currently stopped due to an administrative STOP request.
TRAN	The client is actively engaged in an open transaction. While the connection is in this state, it is considered busy and will not be switched by a nonforced administrative request (see <code>rp_switch</code> on page 195).

Table 6-14 describes the value for each action returned by `rp_who`.

Table 6-14: Values for actions

Value	Description
DEADLOCK	Indicates that a “deadlock” message is pending delivery to the client. The message is sent as soon as the client is in a state in which it can receive the message. This action is usually set due to a call to <code>rp_switch</code> with a value of 1, or when the grace period has expired.
NORETRY	Internal state. This should only appear in conjunction with the SWITCH action. It indicates that, should the switch fail for any reason, the connection is removed from OpenSwitch without attempting to switch to another server.
SERVER	Indicates that the connection is requesting the name of a remote server from a coordination module, to be used either to log in to or to switch to.

Value	Description
STIME	Set in conjunction with SWITCH, this indicates that a grace period has been specified.
STOP	Indicates that the connection is stopped or will stop as soon as possible due to a call to <code>rp_stop</code> .
SWITCH	The remote connection is being switched to another server. The next column indicates the name of the server being switched to.

See also `rp_dump`

This chapter describes notification registered procedures.

Topic	Page
Introduction	205
Using notifications	205
Notification registered procedures	208

Introduction

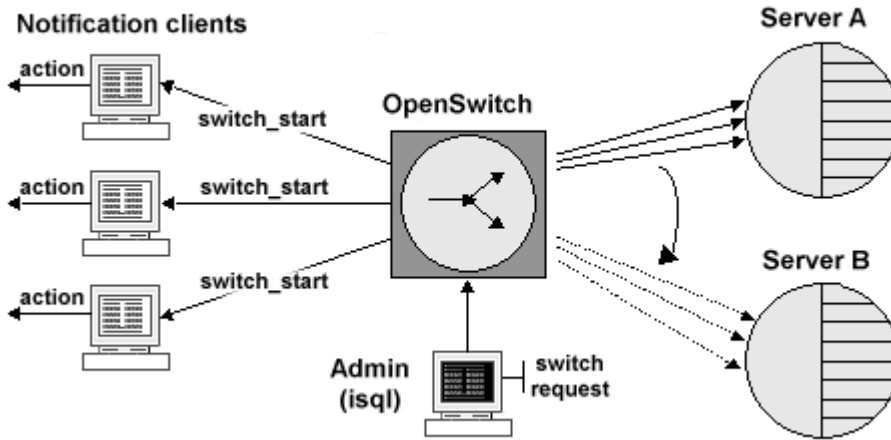
A notification is a special registered procedure that has no associated action or code, but can be used to notify Open Client applications when certain events occur within OpenSwitch.

For details on programming for notification procedures, see the *Open Client Client-Library/C Programmers Guide*.

Using notifications

Use notifications to register multiple client applications with OpenSwitch and provide asynchronous event notification within the server. When the client application receives an event notification, it can take appropriate action; for example, send an e-mail message, as illustrated in Figure 7-1 on page 206.

Figure 7-1: Using notifications



OpenSwitch supports these registered procedures to create notification events:

- `np_req_srv` – notifies the client that a connection blocked a request for the name of a remote server.
- `np_switch_start` – notifies the client that the switching process has been manually started via `rp_switch`.
- `np_switch_end` – notifies the client that switching process started via `rp_switch` has ended.

The next section describes each registered procedure.

Notification registered procedures

np_req_srv

Description

`np_req_srv` communicates with a CM and notify it of connections that are waiting for a response. This procedure is used internally by OpenSwitch and cannot be used by clients. See “Usage” on page 209 for more information.

Syntax

```
np_req_srv spid, username, appname, hostname, database, pool,
rsn_code, rsn_text, cur_server, nxt_server
```

Parameters

- `spid` – the OpenSwitch process ID of the requesting connection.

- *username* – the user name of the requesting connection.
- *appname* – the application that established the requesting connection.
- *hostname* – the client machine.
- *database* – the current database context of the connection. If the server request is the result of a new login, this value is NULL.
- *pool* – the pool name in which the connection is being established.
- *rsn_code* – the reason code that describes why the request is blocked.

Table 7-1: *rsn_code* valid values

Value	Description
1	The connection is logging in. <i>cur_server</i> is NULL, and <i>nxt_server</i> indicates the name of the remote server to use according to the pool in which the connection resides.
2	A login retry. This code indicates that a previous attempt to log in to a server has failed, and that another server name is needed to retry. <i>cur_server</i> contains the name of the server that failed, and <i>nxt_server</i> contains the name of the server to use according to the pool in which the connection resides.
3	Connection lost to remote server. This code indicates that a remote server unexpectedly dropped a connection and that the connection should be switched to the next available server. <i>cur_server</i> contains the name of the failed server, and <i>nxt_server</i> contains the name of the next available server within the pool in which the connection resides.
4	Connection failed. This code indicates that a previous attempt to switch to the next available server failed. <i>cur_server</i> contains the name of the failed server, and <i>nxt_server</i> contains the name of the next available server in the pool in which the connection resides.

- *rsn_text* – text description of the *rsn_code* field.
- *cur_server* – the name of the remote server currently being used by the connection, determined by the value of *rsn_code*.
- *nxt_server* – the name of the remote server that the OpenSwitch has chosen for the connection based upon the pool in which the connection resides.

Return values

Example return values for *np_req_srv* are:

Parameter	Example return value
<i>spid</i>	11
<i>username</i>	sa
<i>appname</i>	ctisql

Parameter	Example return value
<i>hostname</i>	rhino
<i>database</i>	NULL
<i>pool</i>	pool1
<i>rsn_code</i>	1
<i>rsn_text</i>	Login attempt
<i>cur_server</i>	NULL
<i>nxt_server</i>	mayura

Usage

- `np_req_srv` is used to communicate with a coordination module to notify it of connections that are awaiting a response. This procedure is used by OpenSwitch internally to indicate that the connection is blocked and is awaiting a response from the coordination module, which can come in the form of a call to `rp_set_srv`, `rp_switch`, or `rp_kill`. Only these registered procedures or a disconnect from the client can wake up a connection waiting for a response.
- `np_req_srv` is only issued if at least one coordination module is attached and the coordination mode is AVAIL, ALWAYS, or ENFORCED. For more information, see the *OpenSwitch Coordination Module Reference Manual*.

See also`rp_kill`, `rp_set_srv`, `rp_switch`

np_switch_start

Description

Notifies the client that the switching process has been manually started via `rp_switch`.

Syntax`np_switch_start`**Usage**

`np_switch_start` is broadcast to all listening clients as soon as `rp_switch` is executed. It indicates nothing about the success or failure of the switching process, only that it has begun.

See also`rp_switch`, `np_switch_end`

np_switch_end

Description

Notifies the client that switching process started via `rp_switch` has ended.

Syntax`np_switch_end`

Usage	<code>np_switch_end</code> is broadcast when <code>rp_switch</code> completes. This does not indicate the success or failure of the procedure, only that it has completed.
See also	<code>rp_switch</code> , <code>np_switch_start</code>

Index

Symbols

- [COMPANION] section 91
 - example 91
- [CONFIG] section 64
 - example 83
- [LIMIT_RESOURCE] section 89
 - example 90
- [POOL] section 86
 - example 89
- [SERVER] section 83
 - example 85

A

- actions for mutually-aware failure types 113
- Adaptive Server
 - connection context 14
 - deadlocks and RPCs 15
 - failover in mutually-aware OpenSwitch servers 109
 - max network packet size** causing failed client connection 30
 - temporary tables and cursors 15
 - using server pools 4
- application
 - connection 2
 - remote 19
- applications, Open Client 205
- attributes, adding to pools 145
- automatic failover 105
- automatically starting an RCM from OpenSwitch 55

B

- balanced mode 6
- bracket wildcard 61
- breaking connections 141

- broadcasting messages to connections 143

C

- caching
 - and state 36
 - connections 34
 - defining 34
 - restoring connections 35
 - setting connection properties 37
 - viewing connection details 36
- cancelling client connection processing 128
- caret wildcard 61
- chained mode 6
- changing
 - pool status 28
 - server status 22
- character sets, international 13
- characters
 - grouping with brackets 61
 - literal 61
 - negating with caret 61
 - negating with escape 61
 - range of, specifying 61
 - single, wildcard for 62
 - string of, specifying 62
- client-side cursor state, monitoring 33
- client-side cursors 39
- CMON_FAIL_ACTION 119
- CMP_FAIL_ACTION 120
- CMs. See coordination modules
- communication state, monitoring 32
- communication states
 - connection 2
 - transaction 2
- configuration
 - dynamic 14
 - mutually-aware OpenSwitch server support 99
 - mutually-aware OpenSwitch server support

- configuration data precedence 107
 - mutually-aware OpenSwitch server support parameters 99
 - mutually-aware OpenSwitch server support table 104
 - OpenSwitch MAX_PACKETSIZE to match **max network packet size** on Adaptive Server 30
 - OpenSwitch to start an RCM automatically 55
 - replication coordination module configuration file
 - additions 57
 - configuration file
 - [COMPANION] section 91
 - [CONFIG] section 64
 - [LIMIT_RESOURCE] section 89
 - [POOL] section 24, 86
 - [SERVER] section 83
 - creating or editing 62
 - editing the OpenSwitch 60
 - empty lines 60
 - false 60
 - off 60
 - on 60
 - OpenSwitch additions 57
 - parameters that cannot be changed using **rp_cfg** 131
 - pound sign (#) 60
 - requirements 60
 - rereading at runtime 130
 - specifying when starting OpenSwitch 59
 - syntax 60
 - true 60
 - using 59
 - using wildcards 60
 - configuration tool 13
 - connections
 - and pools 25
 - attributes, adding 145
 - breaking 141
 - caching 34
 - cancelling client 128
 - context 14
 - defining pools and caching 34
 - establishing an outgoing 29
 - idle client 3
 - killing 141
 - login denied 30
 - managing 28
 - monitoring state 32
 - number of user 16
 - restoring cached 35
 - sending messages to 143
 - server unavailable 30
 - setting caching properties 37
 - state 2
 - state information, dumping 137
 - suspending and resuming 3
 - terminating 3
 - uncached 36
 - viewing cached details 36
 - conventions used in this manual x
 - COORD_STOP sleep event 193
 - coordination modules
 - requirements 9
 - using 9
 - creating a new OpenSwitch configuration file 62
 - cursors
 - client-side 39
 - repositioning 41
 - within dynamic SQL 40
 - cursors, Adaptive Server 15
 - CUSTOM action for mutually-aware mutually-aware
 - OpenSwitch server failures 116
 - custom script example 115
 - CUSTOM_MANUAL action for mutually-aware mutually-aware OpenSwitch server failures 117
- ## D
- database context 32
 - deadlock messages 39
 - deadlocks, Adaptive Server 15
 - debugging
 - disabling messages for 133
 - enabling messages for 133
 - DECLARE CURSOR** 39
 - DEFAULT action for mutually-aware mutually-aware
 - OpenSwitch server failures 114
 - deployment issues 14
 - detecting failures 38
 - direct invocation of registered procedures 127
 - disabling debugging messages 133
 - displaying registered procedures 139

- documentation
 - conventions used in this manual x
 - OpenSwitch online viii
- dumping connection state information 137
- dynamic
 - configuration of OpenSwitch 14
 - SQL support 12
- dynamic SQL
 - cursors within 40
 - execute-immediate method 12
 - failover behavior 41
 - prepare-and-execute method 13

E

- editing the OpenSwitch configuration file 60, 62
- e-mail, result of a notification 205
- empty lines in configuration file 60
- enabling debugging messages 133
- escape wildcard 61
- examples
 - [COMPANION] file 91
 - [CONFIG] file 83
 - [LIMIT_RESOURCE] file 90
 - [POOL] file 89
 - [SERVER] file 85
 - custom script for mutually-aware OpenSwitch
 - server failures 115
 - notification 205
 - np_req_srv** return values 207
- execute-immediate method of executing dynamic SQL 12
- exiting custom and manual scripts 118

F

- failback in mutually-aware OpenSwitch servers 110
- failover
 - Adaptive Server in mutually-aware OpenSwitch 109
 - automatic 105
 - behavior with dynamic SQL 41
- Failover, Sybase 41

- failure type configuration parameters for mutually-aware support 112

failures

- deadlock messages 39
- detecting 38
- failover behavior within dynamic SQL 41
- handling 40
- managing 38

files

- OpenSwitch configuration 60
- forwarding queries and results 32
- FREEZE_CFG_ON_FAIL**
 - and **rp_pool_status** 167, 183
- FREEZE_CFG_ON_FAIL** and **CMP_FAIL_ACTION**
 - configuration parameters association 120
- FULL_PASSTHRU** mode, configuring for OpenSwitch 16, 72

H

- help for registered procedures 139
- high availability, warm standby environment 55

I

- ignore line in configuration file 60
- information, connection state 137
- international character sets 13
- invoking registered procedures 126
- isql**, using registered procedures in 127

J

- jConnect client application deployment update 16

L

- listing available servers 22
- listing available pools 27
- literal characters, specifying 61
- load balancing and chaining 5
- login denied 30

M

- managing
 - connections and threads 28
 - switch requests 33
- managing failures 38
- max network packet size** on Adaptive Server causing failed client connection 30
- MAX_PACKETSIZE
 - problem if Adaptive Server **max network packet size** is set to 512K 73
 - resetting to match **max network packet size** on Adaptive Server 30
- messages
 - deadlock 39
 - debugging 133
 - notifications 12
 - sending to connections 143
- modes
 - balanced 6
 - chained 6
- monitoring
 - client-side cursor state 33
 - communication state 32
 - connection state 32
 - transaction state 33
- monitoring user connections 89
- mutually-aware OpenSwitch servers 8
 - Adaptive Server failover 109
 - CMON_FAIL_ACTION 119
 - CMP_FAIL_ACTION 120
 - configuration data precedence 107
 - configuration parameters 99
 - configuration table 104
 - configuring 99
 - CUSTOM action 116
 - CUSTOM_MANUAL action 117
 - DEFAULT action 114
 - exiting custom and manual scripts 118
 - failback 110
 - failure type actions 113
 - failure type configuration parameters 112
 - invoking manual actions and custom scripts 111
 - NET_FAIL_ACTION 121
 - operations 109
 - requirements 94
 - shutdown 109

- startup 109
- SVR_FAIL_ACTION 122

N

- negating characters 61
- NET_FAIL_ACTION 121
- notifications 12
 - described 205
 - example 205
 - np_req_srv 206
 - np_switch_end 208
 - np_switch_start 208
 - programming for 205
 - summary 206
 - using 206
- number of user connections 16

O

- Open Client applications 205
- OpenSwitch
 - [COMPANION] section in OpenSwitch configuration file 91
 - [CONFIG] section in OpenSwitch configuration file 64
 - [LIMIT_RESOURCE] section in OpenSwitch configuration file 89
 - [POOL] section in OpenSwitch configuration file 86
 - [SERVER] section in OpenSwitch configuration file 83
 - configuration file additions 57
 - configuration tool 13
 - debugging messages 133
 - deployment issues 14
 - dynamic configuration 14
 - dynamic SQL support 12
 - editing the configuration file 60
 - FULL_PASSTHRU mode 16, 72
 - performance 15
 - removing as a service on Windows 47
 - starting 44
 - starting and stopping the RCM from 54

- starting as a service on Windows 45
- using coordination modules 9
- using replication coordination modules 55
- using server pools and routing 4
- operations, mutually-aware OpenSwitch server support 109
- options, setting server 22
- outgoing connections, transparent 2

P

- percent wildcard 62
- performance 15
- pools
 - adding attributes to 145
 - adding servers to 147
 - and connections 25
 - and servers 25
 - balanced mode 6
 - chained mode 6
 - changing status 27, 28
 - defining 23, 34
 - defining in configuration file 24
 - importance of creation order and **rp_pool_create** 155
 - listing available 27
 - load balancing and chaining 5
 - routing and collisions 24
 - state 26
 - using server 4
- pound sign (#) 60
- prepare-and-execute method of executing dynamic SQL 13
- properties
 - setting cached connection 37

Q

- queries and results, forwarding 32

R

- range of characters, specifying 61

- RCMs. See replication coordination modules
- registered procedures
 - described 125
 - help 139
 - invoking 126
 - notifications 205
 - replication coordination module, new 57
 - rp_cancel** 128
 - rp_cfg** 130, 131
 - rp_debug** 133
 - rp_dump** 137
 - rp_go** 137
 - rp_help** 139
 - rp_kill** 141
 - rp_msg** 143
 - rp_pool_addattrib** 145
 - rp_pool_addserver** 147
 - rp_pool_cache** 152
 - rp_pool_create** 154
 - rp_pool_drop** 157
 - rp_pool_help** 158
 - rp_pool_remattrib** 160
 - rp_pool_remserver** 162
 - rp_pool_server_status** 163
 - rp_pool_status** 165
 - rp_rcm_connect_primary** 169
 - rp_rcm_list** 170
 - rp_rcm_shutdown** 171
 - rp_rcm_startup** 172
 - rp_replay** 174
 - rp_rmon** 179
 - rp_server_help** 180
 - rp_server_status** 181
 - rp_set** 185
 - rp_set_srv** 188
 - rp_showquery** 189
 - rp_shutdown** 190
 - rp_start** 191
 - rp_stop** 192
 - rp_switch** 195
 - rp_traceflag** 198
 - rp_version** 200
 - rp_who** 201
 - sp_ps** Open Server 193
 - using in **isql** 127
- related documents viii

- remote server, adding to pool 147
- replication coordination module
 - configuration file additions 57
 - registered procedures, new 57
 - requirements 55
 - starting and stopping from OpenSwitch 54
- repositioning cursors 41
- requirements
 - coordination module 9
 - mutually-aware OpenSwitch server support 94
 - replication coordination module 55
- re-reading configuration file 130
- resource
 - governing 11
 - monitoring limits 89
- restoring cached connections 35
- routing and collisions in pools 24
- routing and server pools 4
- rp_pool_status** registered procedure and
FREEZE_CFG_ON_FAIL 167, 183
- RPC invocation of registered procedures 127
- runtime, rereading configuration file 130

S

- scripts
 - exiting custom and manual 118
 - invoking custom and manual in mutually-aware
OpenSwitch servers 111
- security, enabling SSL 42
- servers
 - adding pools 4
 - adding to pool 147
 - and pools 25
 - changing the status of 22
 - listing available 22
 - setting options 22
 - state 20
 - state changes 21
 - states 20
 - status of 21
 - unavailable for connection 30
- setting
 - connection caching properties 37
- shutdown, mutually-aware OpenSwitch servers 109

- single character wildcard 62
- sp_ps** Open Server registered procedure 193
- SQL
 - cursors within dynamic 40
 - executing dynamic with execute-immediate 12
 - executing dynamic with prepare-and-execute 13
 - failover behavior with dynamic 41
 - forwarding queries and results 32
- SQL, dynamic support 12
- SSL support, enabling 42
- starting
 - and stopping the RCM from OpenSwitch 54
 - mutually-aware OpenSwitch servers 109
 - OpenSwitch 44
 - OpenSwitch as a service on Windows 45
- state
 - and caching 36
 - client-side cursor 33
 - monitoring communication 32
 - monitoring connections 32
 - monitoring transaction 33
- state changes, server 21
- states, server 20
- status
 - changing pool 27, 28
- status, changing server 22
- status, server 21
- string of characters, specifying 62
- summary notifications 206
- suspending and resuming connections 3
- SVR_FAIL_ACTION 122
- switch requests, monitoring 33
- Sybase Failover support 41
- syntax of OpenSwitch configuration file 60

T

- temporary tables, Adaptive Server 15
- terminating connections 3
- threads
 - managing 28
- transaction
 - last 3
 - state 2
- transaction state 2

- transactions
 - monitoring state 33
- transparent outgoing connection 2
- triggers 205
- troubleshooting
 - connection refused when Adaptive Server packet size is set to 512 30
 - login denied 30
 - server unavailable 30

U

- uncached connections 36
- underscore wildcard 62
- updates for jConnect client application deployment 16
- user connections
 - for resource monitoring 89
 - number of 16
- user messaging 12
- using notifications 206

V

- viewing cached connection details 36

W

- wildcards
 - brackets [] 61
 - caret (^) 61
 - escaping 61
 - percent % 62
 - underscore _ 62
- wildcards, using in the configuration file 60

