SYBASE®

Administration Guide

OpenSwitch™

15.1

DOCUMENT ID: DC20191-01-1510-01

LAST REVISED: November 2007

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

Contents

About This Book

Audience

This book is for OpenSwitch™ version 15.1 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 15.1 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, “OpenSwitch Manager,” – describes the OpenSwitch Manager and provides instructions to use it to manage and monitor OpenSwitch servers, Adaptive Servers, pools, coordination modules, and processes.
- Chapter 4, “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 5, “Using the Configuration File” – describes the sections of the configuration file.
- Chapter 6, “Using Mutually-aware OpenSwitch Servers,” describes how to configure and use mutually-aware OpenSwitch servers.
- Chapter 7, “Registered Procedures” – describes how to invoke registered procedures directly or via remote procedure calls, and provides reference pages for each registered procedure.

- Chapter 8, “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

The Sybase OpenSwitch documentation set consists of:

- *OpenSwitch Release Bulletin* – contains last-minute information not documented elsewhere, and descriptions of known problems and available workarounds.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Product Manuals at http://www.sybase.com/support/manuals/.

- *OpenSwitch Installation Guide* – describes system requirements and provides installation and configuration procedures for OpenSwitch software.

- *OpenSwitch New Features Guide* – describes the new and updated features in OpenSwitch.

- *OpenSwitch Administration Guide* (this book) – explains how to administer OpenSwitch and how to configure the product after installation.

- OpenSwitch Manager online help – describes the tasks you can perform in OpenSwitch Manager.

- *OpenSwitch Coordination Module Reference Manual* – describes how to develop and use OpenSwitch coordination modules.

- *OpenSwitch Error Message Guide* – explains how to troubleshoot problems that you may encounter when using OpenSwitch, and provides explanations of error messages.

- *Sybase Software Asset Management User's Guide* – describes Sybase asset management configuration concepts and tasks.

- *FLEXnet Licensing User Guide* – this Macrovision manual explains FLEXnet Licensing for administrators and end users and describes how to use the tools which are part of the standard FLEXnet Licensing distribution kit from Sybase.

- *SAMreport Users Guide* – this Macrovision manual explains how to use SAMreport, a report generator that helps you monitor the usage of applications that use FLEXnet Licensing.

Other sources of information

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

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

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

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

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

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

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

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

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

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

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

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

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

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

Conventions

The following style conventions are used in this manual:

Commands for the C shell, Bash shell, and 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 Bash and 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.
variable	Italic font indicates: • Program variables, such as *myServer* • Parts of input text that must be substituted, for example: *Server*.log • 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: • 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 15.1 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.

CHAPTER 1

Overview

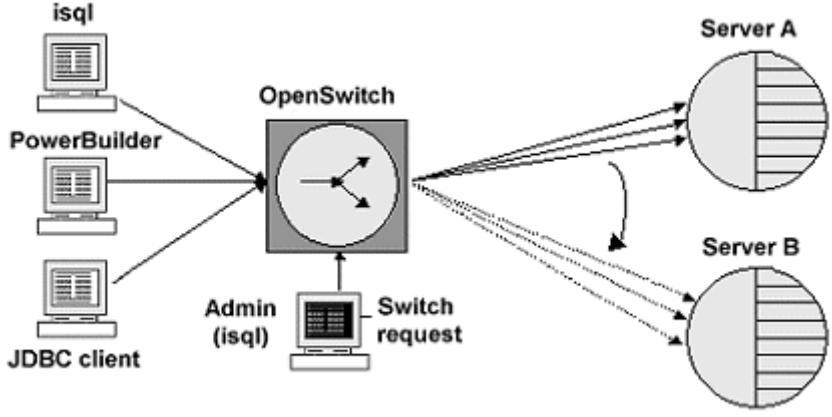
This chapter introduces important concepts about OpenSwitch.

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 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 Adaptive 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.

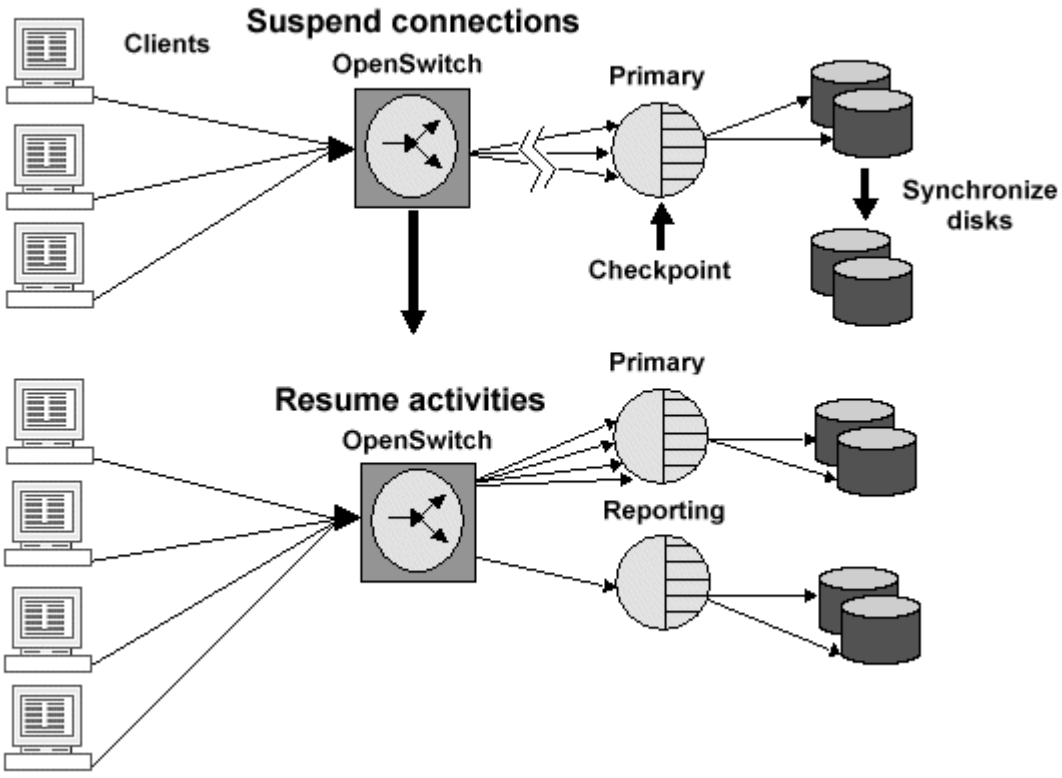
Connection termination

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



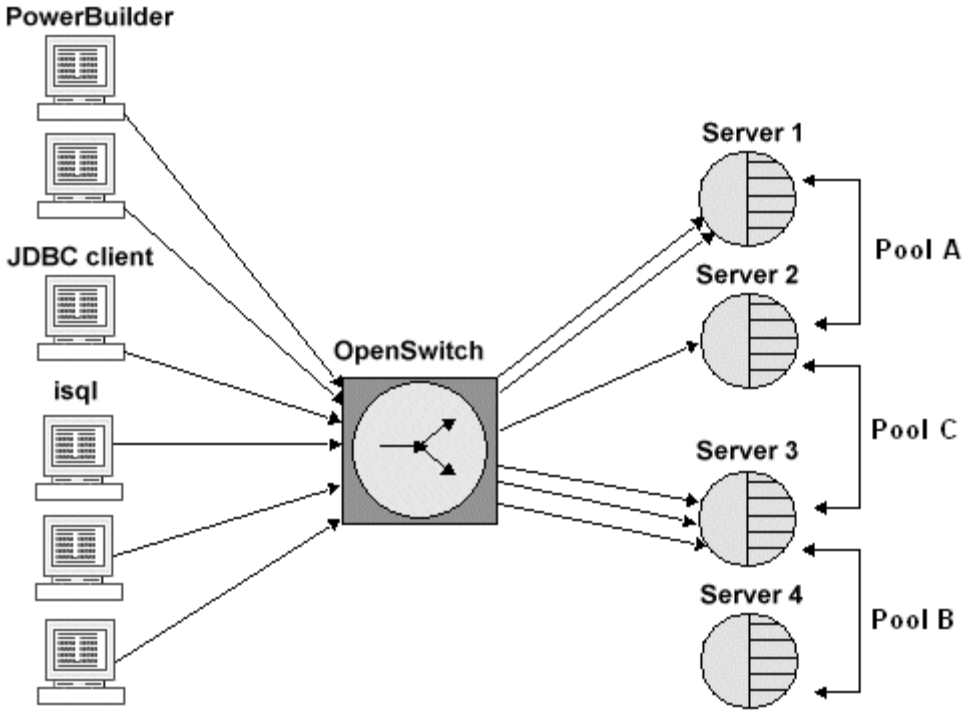
OpenSwitch provides a mechanism to suspend a group of connections (either by pool, server, or individual connection) so that no activity is performed on the server until the connections are resumed.

See rp_stop on page 227 and rp_start on page 225 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.

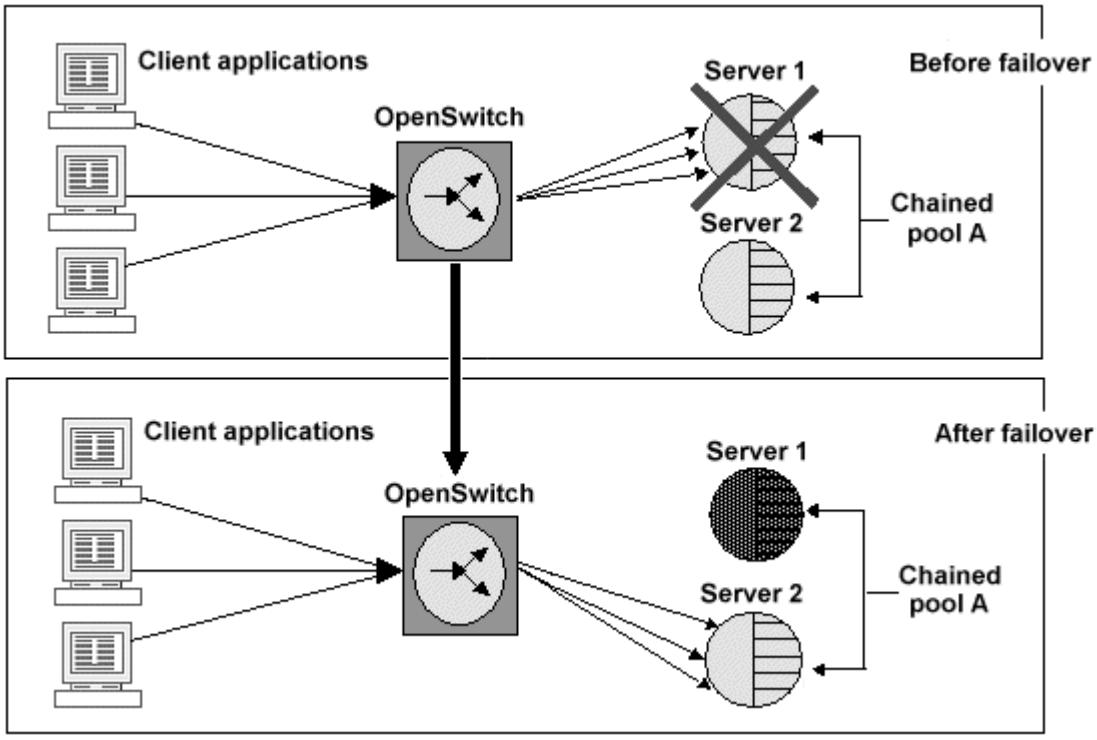
Load balancing and chaining

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.

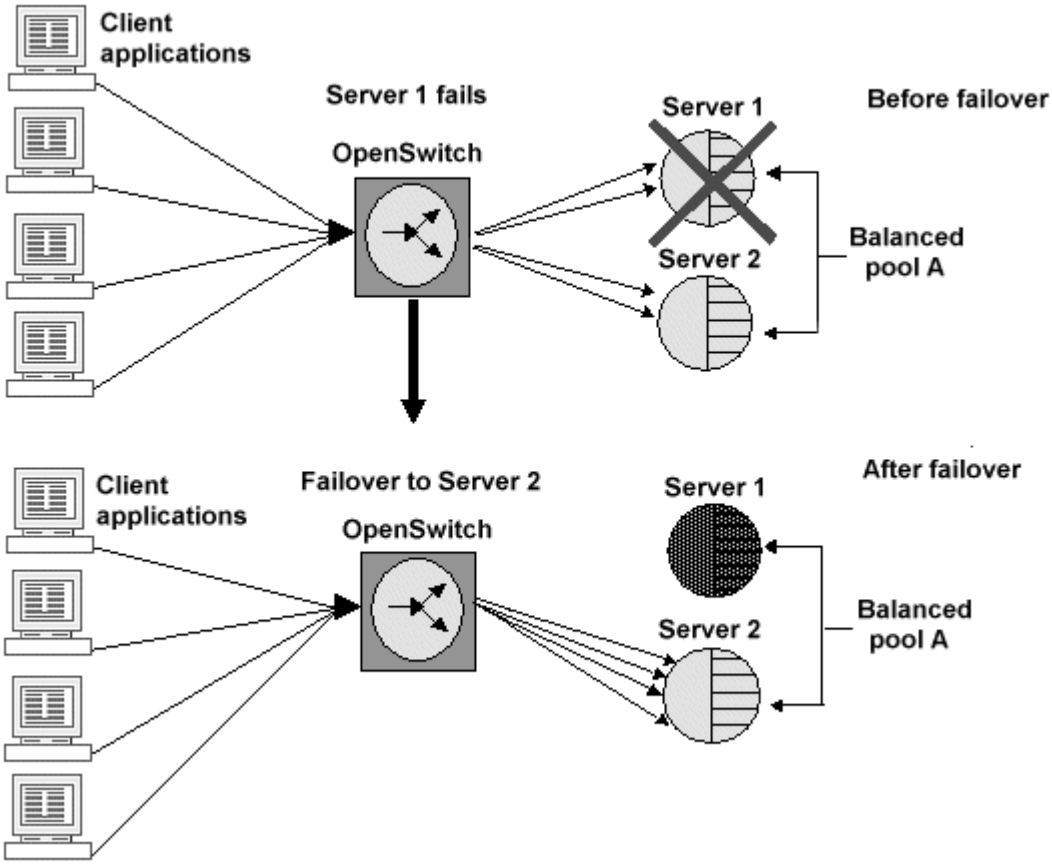
Figure 1-4: Chained mode failover



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.

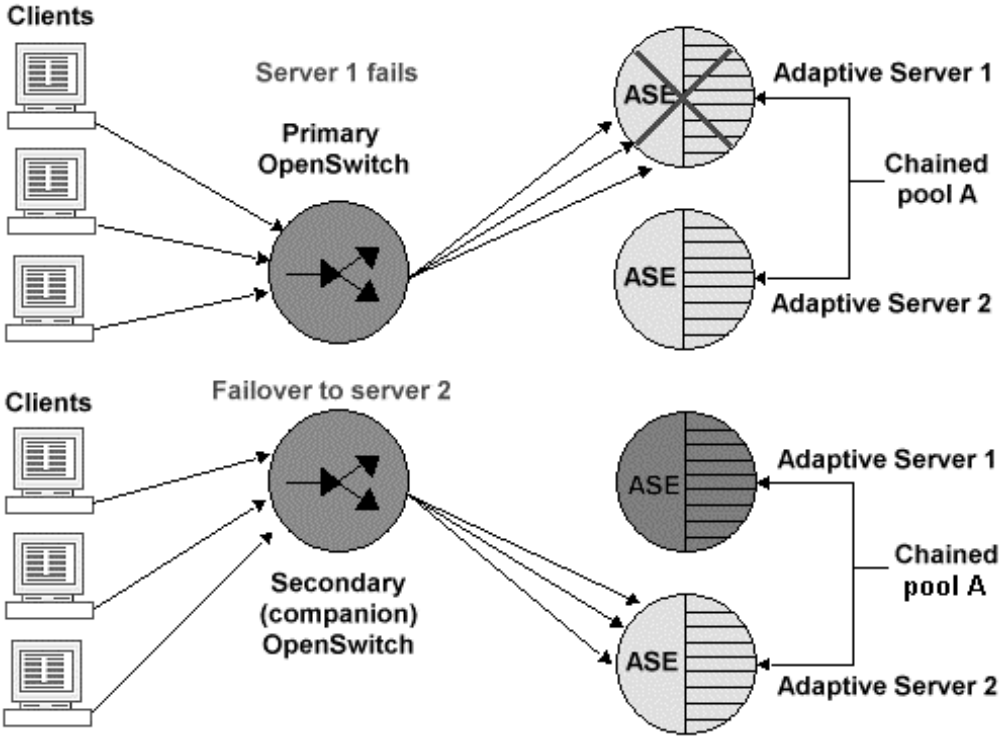
To reduce this overhead, OpenSwitch provides connection pool caching, which allows you to retain connections from specific applications, users, or clients following their disconnection from the client. If the same client reconnects before a specified caching timeout period, the cached connection is reassigned.

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 33.

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 6, “Using Mutually-aware OpenSwitch Servers,” to configure this functionality.

External coordination using coordination modules

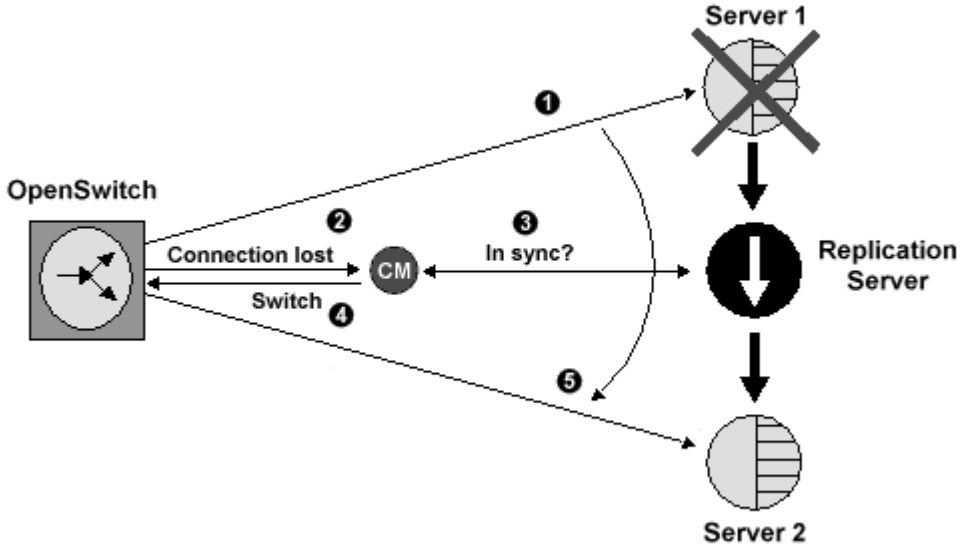
The default behavior of OpenSwitch is to automatically attempt to migrate individual failed connections as they fail. That is, if a single connection fails, it is immediately migrated to the next available server according to the mode of the pool in which the connection resides. However, you may want 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 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* to develop and use 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 88 and “[LIMIT_RESOURCE]” on page 117 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 Chapter 8, “Notification Procedures” for more information.

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 7, “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 *Open Client and Open Server International Developer's Guide* for more information.

Configuration

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

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 Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* for 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 “Using the Configuration File” on page 83 and see the *OpenSwitch Installation Guide* for your platform for more information.

OpenSwitch Manager

OpenSwitch 15.1 and later includes OpenSwitch Manager (OSWM), which is a utility you can use to manage and monitor the OpenSwitch environment. See Chapter 3, “OpenSwitch Manager” for more information.

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 connection context information such as current database context and transaction state, which have 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 See "[CONFIG]" on page 88 to configure "full pass-thru mode" using the *FULL_PASSTHRU* OpenSwitch configuration parameter.

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. See "[CONFIG]" on page 88 for more information.

CHAPTER 2

Concepts and Procedures

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

Configuring OpenSwitch

To define Adaptive Servers and pools in your OpenSwitch environment, you can:

- Use the configuration tool. Access the configuration tool either directly from the OpenSwitch installation program, or by starting it as a standalone application after installation. See Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* for more information.
- Use the OpenSwitch Manager. See Chapter 3, “OpenSwitch Manager” for more information.
- Manually edit the OpenSwitch configuration file. See Chapter 5, “Using the Configuration File” for more information.

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, 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.

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.

State	Description
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.
PRE_LOCKED	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. **Warning!** Do not manually set a server's status to PRE_LOCKED.
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.

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 214 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 Chapter 1, "System Procedures" in the *Adaptive Server Enterprise Reference Manual: Procedures* for instructions on using the sp_dboption system procedure. 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 server 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 177 for more information.

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 179.
- Call rp_pool_server_status to set a server’s status within a pool. See rp_pool_server_status on page 195.
- Specify a server’s status in the [POOL] section of the OpenSwitch configuration file. See “[POOL]” on page 113.

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	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. **Warning!** Do not manually set a pool's status to PRE_UP.
UP	The pool is immediately available for use.
PRE_DOWN	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. **Warning!** Do not manually set a pool's status to PRE_DOWN.
DOWN	The pool is unavailable, and is not considered for use by any new client connections established to OpenSwitch.
PRE_LOCKED	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. **Warning!** Do not manually set a pool's status to PRE_LOCKED.
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.

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 139 for more information. The pool blocks on all new connections until rp_go 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 190 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:

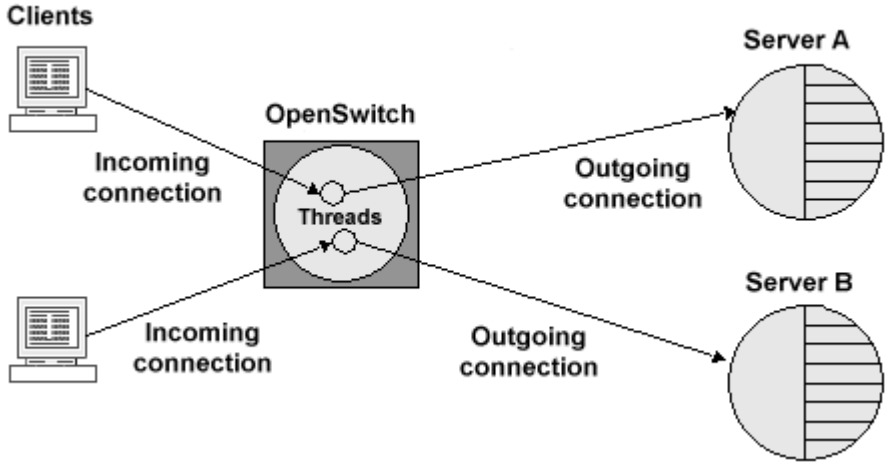
```
pool_name  status
---------- --------
POOL_A     LOCKED
```

See rp_pool_status on page 198 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%\ini\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 set the Adaptive Server max network packet size to 2048, or you can set the OpenSwitch `MAX_PACKETSIZE` to 512.

❖ **Setting Adaptive Server *max network packet* size to 2048**

1 Set the SYBASE environment variable at a command prompt:

- UNIX – in *$SYBASE* enter:

  ```
  SYBASE.csh
  ```

- Windows – in *%SYBASE%* enter:

  ```
  SYBASE.bat
  ```

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.

❖ **Setting OpenSwitch MAX_PACKETSIZE to 512**

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 at a command prompt:

- On UNIX – in *$SYBASE* enter:

```
SYBASE.csh
```

- On Windows – in *%SYBASE%* enter:

```
SYBASE.bat
```

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

```
isql -UAdministrator_UserName
     -PAdministrator_Password
     -SOpenSwitch_ServerName
```

c Execute:

```
rp_shutdown
go
```

2 Use a text editor to modify the OpenSwitch configuration file *OpenSwitch_ServerName.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 either using the GUI configuration tool during or after installation, by using OpenSwitch Manager, or by manually editing the OpenSwitch configuration file. You specify *CACHE* values by manually editing the OpenSwitch configuration file.

See Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* and Chapter 5, “Using the Configuration File” in this book for more information.

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 the *CACHE* duration of a pool 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 184 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 mean 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 Server 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 237 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 169 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 more information.
- 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 immediately. 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.15.0.3
```

See “Security features,” in Chapter 2, “Client-Library Topics” in the *Open Client Client-Library/C Reference Manual* for more information.

CHAPTER 3

OpenSwitch Manager

This chapter describes how to use Sybase Central™ and the OpenSwitch Manager to manage OpenSwitch servers. Sybase integrates its systems management tools into one desktop product, called Sybase Central. You can manage each server product, such as OpenSwitch, Replication Server, or Adaptive Server, from Sybase Central.

Sybase Central and OpenSwitch Manager are options you can install as part of your OpenSwitch installation. You must install Sybase Central if you want to use OpenSwitch Manager. See the *OpenSwitch Installation Guide* for your platform for installation instructions.

Understanding OpenSwitch

It is important for you to understand OpenSwitch concepts before you configure your OpenSwitch environment using OpenSwitch Manager. See the *OpenSwitch Administration Guide* to help you learn about OpenSwitch.

Sybase Central

Sybase Central is a graphical management tool for Sybase products. It implements the Sybase enterprise management strategy, which calls for a single management console, seamlessly integrated across all server and middleware products. It connects to and manages Sybase products that are running on any Sybase-supported platform. OpenSwitch Manager is a *plug-in* to Sybase Central.

The OpenSwitch installation process installs the OpenSwitch Manager plug-in and registers the plug-in in Sybase Central. When you start Sybase Central, you can see the OpenSwitch Manager plug-in icon below the Sybase Central icon.

Windows

Use any of these methods for your Windows operating system to start Sybase Central:

- Select Start | Programs | Sybase | Sybase Central v4.3.
- Navigate to *%SYBASE%\Shared\Sybase Central 4.3\win32* and execute *scjview.exe*.
- Navigate to *%SYBASE%\OSWP\bin* and execute *oswplugin.bat*.

UNIX

Use any of these methods for your UNIX operating system to start Sybase Central:

- Navigate to *$SYBASE/shared/sybcentral43* and execute *scjview.sh*.
- Navigate to *$SYBASE/OSWP/bin* and execute *oswplugin*.

You can set Sybase Central options to configure how it displays and to enable the Fast Launcher which reduces the time Sybase Central takes to start up. Select Tools | Options to set these options.

See the Sybase Central online help for more information. Select Help | Sybase Central or press the F1 key to access the online help.

To view the activities of plug-ins that you register in Sybase Central, select Tools | Log Viewer.

OpenSwitch Manager features

OpenSwitch Manager (OSWM) is a management utility you can use to develop, manage, and monitor an OpenSwitch environment that includes OpenSwitch servers, Adaptive Servers, coordination modules (CM), and replication coordination modules (RCM) that participate in the environment.

With its easy-to-use interface, OSWM allows you to perform many administrative tasks for which you would otherwise use commands and registered procedures, including:

- Adding, assigning, altering, and deleting OpenSwitch objects, such as OpenSwitch servers, Adaptive Servers, pools, and processes.
- Managing, troubleshooting, and monitoring the availability of multiple OpenSwitch servers, Adaptive Servers, and pools, and the state of connections to all servers and pools.
- Controlling the flow of data by suspending, switching, and resuming connections either manually in response to an administrative request, or automatically in response to an Adaptive Server failure.

Use the online help in OSWM to guide you when you perform tasks.

OpenSwitch Manager interface

The Sybase Central window has two panes—Folders pane on the left and Details pane on the right.The OSWM plug-in object displays under the Sybase Central object. All other objects that belong to the OpenSwitch environment that the OSWM manages, display in a tree structure below OSWM within the Folders pane:

- Server groups.
- OpenSwitch servers – display with their port numbers, and in braces the user ID that the administrator uses to connect to them.
- Folders – Pools, Resource Governor, Adaptive Servers, Coordination Modules, and Processes.

The Details pane displays the contents of each server group, folder, the status of OpenSwitch servers and objects, and links to windows to add server groups, servers, and pools. You can also view information on each object in the Status bar at the bottom of the Sybase Central window.

Besides the main menu, you can access most of the functions in OSWM either by clicking the relevant button in the toolbar or by right-clicking to access a context menu.

Editing OpenSwitch Manager properties

You can change properties of OSWM such as, the timeout period for connecting and the *Processes* folder refresh interval.

❖ **Editing OpenSwitch Manager properties**

1 Select the OpenSwitch Manager, then select File | Properties. You can also right-click OpenSwitch Manager and select Properties, or you can select the Properties button from the toolbar.

2 Select the Preferences tab. You can change:

- Login Timeout – the time in seconds that you can set for OSWM to try to connect to a particular server. The default is 30 seconds.
- Process Folder Refresh Interval – the interval in seconds when OSWM next refreshes the process folder to display the latest status of processes such as client connections, coordination modules, and replication coordination modules that are connected to OpenSwitch currently. The default is 10 seconds.

3 When you finish making changes, click Apply, then OK. You can also cancel any changes you make, or restore the default settings.

Managing server groups

You can organize OpenSwitch servers into groups such as by geography or function. You can execute registered procedures, or connect or disconnect servers in the server group, so that you do not have to issue the same command to each server individually.

For example, you can add the primary and secondary OpenSwitch servers that form a mutually-aware cluster to a server group. This makes it easier to issue commands that must be issued to both servers in a mutually-aware cluster.

❖ Creating server groups in OpenSwitch Manager

Note The Default server group is created automatically when you register the OSWM plug-in.

1 Select OpenSwitch Manager, then select File | New | Server Group. You can also double-click the Add Server Group icon in the Details pane on the right side of the Sybase Central window.

2 Provide a name and a description for the new server group and click Next.

3 Select a server from "Add Server Group - Add Servers" window. These are servers listed in the *interfaces* file in UNIX or the *sql.ini* file in Windows. You can also see a list of these servers if you select the OpenSwitch Manager icon, and the select the Servers tab in the Details pane of Sybase Central.

4 Provide the user name and password in the Connect to OpenSwitch Server window, and then click OK to add the server to the server group.
If the OpenSwitch server you want to connect to is not listed, click Finish, and then right-click the server group and select Connect. See "Connecting OpenSwitch servers" on page 48 to connect to your OpenSwitch server.

5 Click Next to display a summary of the information you provided for the new server group, and then click Finish to save the information.

6 Select the server you added to the server group, and then select File | Connect to connect to the server.

❖ Deleting server groups

You can only delete a server group after you disconnect all servers from the group. See "Disconnecting all servers in a server group" on page 48.

1 Select the server group you want to delete.

2 Select Edit | Delete, and then click Yes.

Note You cannot delete the Default server group.

❖ Editing server group properties

1 Select a server group, and then select File | Properties.

2 Edit the server name and edit or add text in the Comments field.

3 Click OK to save your changes.

Note You cannot edit the properties of the Default server group.

❖ **Disconnecting all servers in a server group**

You can disconnect all servers attached to a server group.

1 Select a server group.

2 Select File | Disconnect All. All the OpenSwitch servers in the server group disconnect and their icons are greyed out.

❖ **Reconnecting all servers in a server group**

You can reconnect to all the servers in a server group you previously disconnected.

1 Select a server group with disconnected servers. The icon for a disconnected server is greyed out

2 Select File | Connect All.

Connecting and disconnecting OpenSwitch servers

Administrators can use OSWM to connect to OpenSwitch servers which are already running or to disconnect OpenSwitch servers from OSWM. Once you connect to an OpenSwitch server, you can manage the server and other resources with OSWM.

You can disconnect, remove, or reconnect OpenSwitch servers from OSWM. Disconnecting or removing a server does not shut the server down, but you cannot use OSWM to manage it.

❖ **Connecting OpenSwitch servers**

1 Select a server group. If you are connecting to a server while creating a server group or connection profile, follow step 3 onwards.

2 Select File | Connect. You can also right-click a server group and select Connect, or select the Connect button from the toolbar.

3 Provide the user name and password that allows you to connect to an OpenSwitch server.

4 In the Server Name field, provide the host name and port number of the OpenSwitch server you want to connect to, in this format:

```
host name:port number
```

You can also select a different OpenSwitch server from the Server Name field.

5 Select OK to return to the Sybase Central main window. The OpenSwitch server that you connect to appears under the server group you selected earlier.
If you select the server group, you can see columns that list the properties of each OpenSwitch server in the Details pane:

- Server
- Version
- Platform/OS
- Host:Port
- Status – whether the server is currently running.
- Connected – whether the server is currently connected to OSWM.

❖ **Using connection profiles to connect OpenSwitch servers.**

You can use the Connection Profiles feature of Sybase Central to add and automatically connect to servers in the Default server group when you start Sybase Central.

1 Select Tools | Connection Profiles.

2 Click New to display the New Profile window.

3 Provide a name for the profile.

4 Select the New Profile option.

5 Verify that OpenSwitch is selected in the Plug-in field.
You can select the other options such as “Create a profile so that all users can access it” and “Copy Profile”. Click the Help button for more information on these options.

6 Click OK to launch the Connect to OpenSwitch Server window. See Connecting OpenSwitch servers to connect the server.

❖ **Disconnecting OpenSwitch servers**

1 Select the OpenSwitch server you want to disconnect.

2 Select File | Disconnect. You can also right-click the server and select Disconnect, or select the Disconnect button from the toolbar. The OpenSwitch server disconnects and its icon is greyed out.

❖ **Reconnecting OpenSwitch servers**

1 Select a disconnected OpenSwitch server. The icon representing a disconnected server is greyed out.

2 Select File | Connect. You can also right-click the disconnected server and select Connect, or select the Connect button from the toolbar.

❖ **Removing OpenSwitch servers from a server group**

You can remove an OpenSwitch server from OSWM.

1 Select a disconnected OpenSwitch server.

2 Right-click and select Remove to remove the OpenSwitch server.

3 Select Yes in the Confirm Delete window.

Note If you use OSWM to remove an OpenSwitch server, you cannot use OSWM to administer the server or change its configuration settings. You must login and connect to the server again.

Configuring OpenSwitch servers

You can use OSWM to view and edit the server configuration settings. These settings affect the behavior of your OpenSwitch server when it starts and when it runs.

These are the same settings that are stored in the OpenSwitch configuration file when you use the configuration tool during and after installation. See Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* for your platform for more information.

You can also manually edit the configuration file to change the OpenSwitch server configuration settings. See Chapter 5, “Using the Configuration File,” for more information.

❖ **Viewing OpenSwitch server properties**

1 Select an OpenSwitch server in the Sybase Central window.

2 Select File | Properties to display the OpenSwitch Server Properties window. The host name and port number of the server, and the user name that you use to connect to the server, display in the title bar of the Properties window. You can also right-click the OpenSwitch server and select Properties, or select the Properties icon from the tool bar.

3 Select the General tab to display information about the OpenSwitch server such as version, release number, operating system, build number and date, and whether the server is configured for mutually-aware or high availability support.

❖ **Editing OpenSwitch server properties**

You can use OSWM to edit OpenSwitch server properties such as, configuration options, debug message settings, and trace flag settings.

1 Select an OpenSwitch server in the Sybase Central window.

2 Select File | Properties.
You can also right-click the OpenSwitch server and select Properties, or select the Properties button from the toolbar. You can also right-click the OpenSwitch server and select Configure to go directly to the Configuration tab.

3 Select the:

- Configuration tab – to set the server configuration parameters. Double-click the value you want to set and provide a value. See Table 5-1 on page 89 for a description of each configuration option.
- Debug Messages tab – to set debug flags that enable different debug actions and messages specific to OpenSwitch. Double-click the state you want to set and select On or Off. See Table 4-2 on page 75 for a description of each debug flag.
- Trace Flag tab – to set trace flags to enable Open Server-level debugging messages. Double-click the state you want to set and select On or Off. See Table 4-1 on page 74 for a description of each trace flag.

4 Select Apply and OK to save your changes.

Setting up the OpenSwitch environment

You can use OSWM to define pools and add Adaptive Servers to pools.

In Sybase Central, under each OpenSwitch server, there are objects called *folders* that contain the resources you can manage with OSWM:

- Pools – contains the different Adaptive Server pools that you can define and maintain in OpenSwitch.

  Within each pool, there is an "ASE server" folder that contains the Adaptive Servers you assign to that pool.
- Resource Governor – contains information on the limits that apply to the username, hostname, and application name of incoming client connections.
- Coordination Module – contains either the Coordination Modules or Replication Coordination Modules connected to the OpenSwitch server.
- Processes – contains information on the client connections connected to the Adaptive Servers that are within the OpenSwitch environment.

Managing pools

A pool is a group of Adaptive Servers that you define within the OpenSwitch environment. A pool can contain one or more servers that OpenSwitch treats as a self-contained failover group, so all connections within the group fail over only to servers you define within the group. See "Defining pools" on page 23 for more information.

A pool can optionally define the set of connections or connection attributes that it manages. You can use the user name, application name, or connection type to define this association between a connection and a pool.

❖ **Creating pools in OpenSwitch Manager**

Create pools using OSWM, or by:

- Using the OpenSwitch configuration tool during and after installation.
- Manually editing the OpenSwitch configuration file. See "Defining a pool" on page 24.

- Using the rp_pool_create registered procedure. See rp_pool_create on page 186.

1 Select the OpenSwitch server for which you want to create a pool.

2 Select File | New | Pool. You can also select the *Pools* folder and double-click the Add Pool icon in the Details pane on the right side of the Sybase Central window.

3 Provide the name of the new pool in an empty row of the Pool Name column. Select Pool | Add or click the Add (+) button in the toolbar to create a new row.

4 Select the status for the new pool: UP, DOWN, or LOCKED. Sybase recommends that you select the DOWN state for a new pool so that processes do not switch to the new pool immediately. Change the pool status to UP or LOCKED after you assign Adaptive Servers to the pools and verify the pool is ready to accept connections. See Table 7-7 on page 187 for detailed descriptions of pool states.

5 Select the new pool and select Pool | Properties to display the Pool Parameters window. You can also right-click the pool and select Properties, or select the Properties button from the toolbar.

6 Select the Parameters tab. Select:

   - Relative Pool – to create the new pool relative to an existing pool. Select the existing pool.
   - Pool Position – to specify the position of the new pool relative to the existing pool: HEAD, TAIL, BEFORE, and AFTER. See Table 7-6 on page 186 for more information on these options.
   - Pool Mode – to provide the routing and switching modes for the new pool: CHAINED or BALANCED. See Table 7-8 on page 187 for more information on these modes.

Warning! Be careful when you choose the pool parameters. You cannot change the pool parameters after you save the new pool.

7 Click OK to return to the Pool Editor window.

8 Select Pool | Add in the Pool Editor window to create additional pools. Follow step 3 to step 7 for each pool you want to create.

9 Select File | Save Pool to save the pool and not exit the Pool Editor window. To save the pool and exit the editor, select File | Save Pool and Close Editor.

After you create and save pools, you can assign pool connection attributes to each pool. See Adding pool attributes.

❖ **Adding pool attributes**

You can assign attributes to a pool that apply to all Adaptive Servers in the pool. You can assign pool connection attributes only to an existing pool.

1 Select the pool to which you want to add attributes.

2 Select File | Properties.You can also right-click the pool and select Properties, or select the Properties button from the toolbar.

3 Select the Connection Attributes tab.

4 Click the Add button to the right of the tab.

5 Select an attribute from the Attribute field and specify its value in the Value field:

- *username* – to restrict access to connections initiated with the user name you specify.
- *appname* – to restrict access to connections from the application you specify.
- *hostname* – to restrict access to connections from the host you specify.

6 Select Apply and OK to save the connection attributes you specify.

You can also change or remove the pool connection attributes in the pool Properties window.

❖ **Viewing pool activity and status**

You can use the Pool Properties window to view the pool configuration, pool status, and current activities that have an impact on the pool.

1 Select a pool.

2 Select File | Properties. You can also right-click the pool and select Properties, or select the Properties button from the toolbar.

3 Select the General tab to view:

- Status – UP, DOWN, or LOCKED. See Table 7-7 on page 187 for more information.
- Routing Mode of Pool – BALANCED or CHAINED. See Table 7-8 on page 187 for more information.

- Adaptive Server Assigned to Pool – lists all the Adaptive Servers that you assign to this pool.
- Processes Blocked on a Locked Pool – number of client connections that are currently blocked on a LOCKED pool.
- Cache Interval (seconds) – number of seconds that an outgoing connection is maintained after a client, application, or user disconnects from an Adaptive Server. See "Connection pools and caching" on page 7 and "Using connection caching" on page 33 for more information.

4 Select the Connection Attributes tab to add pool attributes. See "Adding pool attributes" on page 54 for more information.

5 Select the Server tab to view the status of the Adaptive Servers that you have configured for the pool. See "Server state" on page 20 for more information.

6 Select the Pool Status tab to view or change the pool status in the Pool Status field—UP, DOWN, or LOCKED.

❖ Deleting pools

You can delete pools from the main Sybase Central window or the Pool Editor window.

Warning! You cannot delete a pool that contains Adaptive Servers. The servers in the pool may have processes that are still running. If you attempt to delete a pool that contains Adaptive Servers, you see an error message.

Only after you verify there are no processes that are connected to the Adaptive Servers in the pool, can you remove the Adaptive Servers from the pool and delete the pool.

---

1 Select the pool you want to delete.

2 Select File | Properties. You can also right-click the pool and select Properties, or select the Properties button from the toolbar.

3 Select the Server tab and verify that there are no servers listed. If there are any servers, you cannot delete the pool.

4 Click OK to save your changes and return to the Sybase Central window.

5 Select Edit | Delete to display the Confirm Delete window.

6 Select Yes to confirm you want to delete the pool.

Configuring Adaptive Servers for OpenSwitch

You can add Adaptive Servers to pools and configure the servers for your OpenSwitch environment.

You can also use the rp_pool_addserver registered procedure to add and configure Adaptive Servers for a pool. See rp_pool_addserver on page 179.

❖ **Adding Adaptive Servers to a pool**

1 Select the pool to which you want to add Adaptive Servers.

2 Select File | New | Server. You can also double-click the Add Adaptive Server icon in the Details pane.

3 Provide the name of the new server in an empty row of the Server Name column. Select Server | Add from the menu bar or click the Add (+) button in the toolbar to create a new row.

4 Select the status for the new server—UP, DOWN, or LOCKED. See "Server state" on page 20 for a description of each state. Sybase recommends that you select the DOWN state so that processes do not switch to the new server immediately. You can change the Adaptive Server status to UP or LOCKED after you assign Adaptive Servers to the pools and verify the pool is ready to accept connections.

5 Select the new server, then select Server | Properties.

6 Click the Adaptive Server Attributes tab. Select:

- Relative Adaptive Server – to create the new server relative to an existing server. Select the existing server.
- Position – to position the new server relative to the existing server: HEAD, TAIL, BEFORE, or AFTER. See Table 7-5 on page 181 for descriptions of each position.

Warning! Be careful when you choose the server attributes. You cannot change the server attributes after you save your new server.

7 Click OK to return to the Adaptive Server window for your pool.

8 Select Server | Add to add additional servers. Follow step 3 to step 7 for each server you want to add to the pool.

9 Select File | Save Adaptive Server to save the server and not exit the Adaptive Server Editor window. To save the server and exit the Adaptive Server Editor window, select File | Save Adaptive Server and Close Editor.

❖ **Viewing server properties**

1 Select a server.

2 Select File | Properties.

3 Select the General tab to view these properties:

- Status – UP, DOWN, or LOCKED. See "Server state" on page 20 for more information.
- High Availability Type – whether the OpenSwitch server is configured for high availability support.
- Mutually Aware Cluster Table – whether the OpenSwitch server is configured for mutually-aware support. This field applies only if the OpenSwitch server is part of a mutually-aware cluster.
- Connection Monitor User Name – the user name that the connection monitor (CMON) uses to connect to the Adaptive Server.

❖ **Deleting Adaptive Servers from a pool**

You can delete Adaptive Servers from a pool. You must verify that there are no user connections for the Adaptive Server and that the server status is DOWN before you delete the server from the pool.

You can also use the rp_pool_remsever registered procedure to delete the server from the pool. See rp_pool_remserver on page 194 for more information.

When you delete the server from the pool, the server still runs but does not accept user connections assigned through the pool.

1 Select the Adaptive Server you want to delete from the pool.

2 Select Edit | Delete. The Confirm Delete window displays.

3 Select Yes to delete the server from the pool.

Managing the OpenSwitch environment

You can use OSWM to execute registered procedures on a server groups, resume OpenSwitch servers after a failure, and manage and monitor coordination modules, processes, Adaptive Servers, and resources, such as OpenSwitch server hosts and applications.

Executing registered procedures on a server group

You can execute registered procedures on all servers in the server group.

❖ **Executing registered procedures on a server group**

1. Select a server group, and then select File | Execute Registered Procedure.
2. Select the servers where you want to execute registered procedurea. You can select the Options button to set display options for the registered procedures output display format at the bottom of the Server Group Registered Procedure Execution window.
3. Enter registered procedures in the Registered Procedure Statements field. See Chapter 7, “Registered Procedures” for descriptions of registered procedures.
   You can enter and execute multiple registered procedures as a batch. For example:

   ```
   rp_who
   go
   rp_server_status
   go
   ```

4. Click Execute to run the registered procedure on the servers you selected.
5. Select the server tab at the bottom of the Server Group Registered Procedure Execution window, and then select one of the registered procedures from the list of registered procedures you executed, to view the output of that registered procedure for the server you selected.

6 Select the Result Set tab at the bottom of the Server Group Registered Procedure Execution window, to view the detailed output for the registered procedure you executed on the server you selected, and then select the Messages tab to view or copy messages generated during the execution of the registered procedure that may help resolve any problems.
If you execute the registered procedure successfully, the Status column for the server at the top of the Server Group Registered Procedure Execution window displays “Successful”.

Resuming OpenSwitch servers

In a mutually-aware support environment, you can resume an OpenSwitch server when the server is waiting for intervention from the administrator after it performs any of these manual failure actions—companion, connection monitor, network, or Adaptive Server.

Note You cannot use the Resume OpenSwitch Server option if your servers are not part of a mutually-aware support environment. You cannot resume an OpenSwitch server which has not performed a manual failure action and which is still running.

See “Invoking custom and manual scripts” on page 139 for more information.

❖ **Resuming OpenSwitch server after companion, connection monitor, or network failure**

1 Select the OpenSwitch server you want to resume.

2 Select File | Resume OpenSwitch Server.

❖ **Resuming OpenSwitch server after Adaptive Server failure**

Resume the OpenSwitch server when the primary Adaptive Server status is DOWN and after the OpenSwitch server performs the manual server failure action.

1 Select the Adaptive Server that belongs to a pool of the OpenSwitch server you want to resume.

2 Select File | Resume OpenSwitch Server.

Changing Adaptive Server status

You can use OSWM to change the state of Adaptive Servers connected to the OpenSwitch server. The state indicates whether the Adaptive Server is available to connections and processes.

❖ **Changing Adaptive Server status**

1. Select a server.
2. Select File | Properties.
3. Select the Adaptive Server Status tab.
4. Select UP, DOWN, or LOCKED. See “Server state” on page 20 for a description of each state.
5. Select OK to save your changes.

Managing connections to Adaptive Server

❖ **Stopping connections to Adaptive Server**

1. Select the Adaptive Server where you want to stop connections.
2. Select File | Connections | Stop.
3. You can select these options:
   - Ignore Transaction State When Stopping a Connection – OpenSwitch stops all connections to the Adaptive Server whether or not they are involved in an open transaction.

     If you do not select this option, OpenSwitch stops connections as soon as they complete their current transaction.
   - Ignore Remote Adaptive Server Failures – if you select this option, and if an Adaptive Server actively being used by a stopped connection fails, the normal failover process proceeds for the connection as soon as you restart the connection.

     If you do not select this option, and 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 you restart the connection.

You can also use the rp_stop registered procedure to stop connections. See rp_stop on page 227.

4 Click OK to save your options and stop all connections to the Adaptive Server.

❖ **Switching connections from Adaptive Server**

You can switch all connections from one Adaptive Server to another Adaptive Server in the same pool. You cannot choose to which server you want to switch connections. Connections switch to servers according to the server order you specify when you add servers to pools. See "Adding Adaptive Servers to a pool" on page 56.

1 Select the Adaptive Server where you want to change the connections.

2 Select File | Connections | Switch.

3 You can select these options:

- Grace Period (in seconds) – to specify the maximum number of seconds to wait before forcefully switching busy connections. A value of 0 indicates that OpenSwitch does not enforce a grace period.
- Force switch – if you select this option, OpenSwitch forcefully switches all connections, even if the connections are currently busy, they are either actively in the middle of communicating with a remote server, or in the middle of processing open transactions.

  If you do not select this option, OpenSwitch switches connections only if they are not busy.

You can also use the rp_switch registered procedure to switch connections. See rp_switch on page 230.

4 Click OK to save your options and switch all connections from the Adaptive Server.

❖ **Starting connections to Adaptive Server**

You can use OSWM to start connections to a server.

1 Select the Adaptive Server where you want to change the connections.

2 Select File | Connections | Start.

3 Click OK to start the end user connections to the server that you previously stopped.

You can also use the rp_start registered procedure to start connections. See rp_start on page 225.

Monitoring resources

You can monitor the resources that connect to Adaptive Servers within the OpenSwitch environment. These are resources that you define in OpenSwitch by user name, host name, or application name.

OpenSwitch allows you to automatically cancel or terminate connections that overuse Adaptive Server, however, you cannot use OSWM to perform these actions. Configure these settings in the OpenSwitch configuration file. See “Resource governing” on page 11 for more information.

❖ **Monitoring resources with the Resource Governor**

1. Select the OpenSwitch server you want to monitor.
2. Select the *Resource Governor* folder to display attributes such as OpenSwitch server hosts and applications.
3. You can view the different values for each attribute:
   - Entry – the name of the attribute.
   - Action – the actions that you can perform on these attributes in OpenSwitch.
   - Active Period – the period of time in seconds after which the actions you specify take effect.

Monitoring Coordination Modules

Either coordination modules or replication coordination modules are connected to a specific OpenSwitch server. A CM and a RCM cannot be connected at the same time to the same OpenSwitch server.

❖ **Monitoring Coordination Modules and Replication Coordination Modules**

1. Select the OpenSwitch server you want to monitor.
2. Select the *Coordination Modules* folder to display all Coordination Modules (CM) or Replication Coordination Modules (RCM) connected to the OpenSwitch server.
3. Select any CM or RCM.

4 Select File | Properties.

CM properties:

- Host Name – the host name of the OpenSwitch server that is connected to the CM.
- Process ID – the ID for the CM process.
- Status – whether the CM is active.

RCM properties:

- Name of the Replication Coordination Module.
- RCM Type – whether the Replication Coordination Module is a primary or secondary coordination module.

Managing and monitoring processes

You can use OSWM to monitor and terminate processes connected to Adaptive Servers within an OpenSwitch environment.

❖ **Monitoring processes**

1 Select the OpenSwitch server you want to monitor.

2 Select the Processes folder to all processes connected to the OpenSwitch server. You can view the Adaptive Servers and pools the processes are connected to.

3 Select a process.

4 Select File | Properties:

5 Select the General tab to view these properties:

- Process ID.
- Type.
- Application type – the application name such as, isql.
- Login – the user name used by the application to log in to Adaptive Server.
- State – the current status of the process.

6 Select the Parameters tab to view these parameters:

- Remote Adaptive Server ID – the process ID (*spid*) of the process.
- Action – pending actions that an administrator initiated or pending actions that arise internally such as Adaptive Server failure: DEADLOCK, NORETRY, SERVER, STIME, STOP, or SWITCH. See Table 7-16 on page 238 for descriptions of each type of action.
- Host Name – OpenSwitch server host name.
- Database – the name of the database that the process is connected to.
- Current Adaptive Server – the Adaptive Server that the process is currently connected to.
- Next Adaptive Server – the Adaptive Server that the process can switch to.
- Pool – the pool that contains the Adaptive Servers that the process connects to currently.

❖ **Terminating processes**

1. Select the OpenSwitch server on which you want to terminate a process.
2. Select the Processes folder then select a process.
3. Select Edit | Delete.
4. Select Yes to confirm and terminate the process.

CHAPTER 4

Starting and Stopping OpenSwitch and RCMs

This chapter describes how to start and stop OpenSwitch.

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 Bash, Bourne or Korn Shell, enter:

```
. ./SYBASE.csh
```

3 Go to *$SYBASE/OpenSwitch-15_1/bin*, and enter the following, where -c specifies the name of OpenSwitch configuration file to be used during start-up and *OpenSwitch_ServerName* is the name of your OpenSwitch configuration file:

```
./OpenSwitch -c .../config/OpenSwitch_ServerName.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 70.

See “Using command line options” on page 72 for a list of options you can use to adjust the behavior of OpenSwitch.

❖ **Stopping OpenSwitch**

1 In a terminal window, connect to isql as an administrator and enter in one line, where *OpenSwitch_ServerName* is the name of the OpenSwitch server you want to stop:

```
isql -UAdministrator_UserName -PAdministrator_Password
 -S OpenSwitch_ServerName
```

2 Enter:

```
rp_shutdown
go
```

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 to configure OpenSwitch after installation, see Chapter 5, “Using the Configuration File,” in this guide, and Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* for more information.

❖ **Starting OpenSwitch from the Start menu**

See Chapter 4, “Post- Installation Tasks” in the *OpenSwitch Installation Guide* to add OpenSwitch to the Windows Start menu. To start OpenSwitch:

- 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_1\bin
```

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

```
start OpenSwitch.bat -c ...\config\OpenSwitch_ServerName.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 70.

❖ **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, connect to isql as an administrator and enter in one line, where *OpenSwitch_ServerName* is the name of the OpenSwitch server you want to stop:

```
isql -UAdministrator_UserName -PAdministrator_Password
 -S OpenSwitch_ServerName
```

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_1\config\OpenSwitch_ServerName.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 select the OpenSwitch server name.

a Select Edit | New | Key, then enter `Parameters` as the key name.

b Select the new Parameters key, select Edit | New | String Value, then enter `Application` as the new string value.

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

```
OpenSwitch installation directory\bin\OpenSwitch.bat
-c OpenSwitch configuration file path
-l OpenSwitch install directory\bin\OpenSwitch_ServerName.log
```

Where:

- *OpenSwitch installation directory* – is the drive and directory in which OpenSwitch is installed; for example:

  `D:\Sybase\OpenSwitch-15_1`

- *OpenSwitch configuration file path* – is the location of the OpenSwitch configuration file; for example:

`D:\Sybase\OpenSwitch-15_1\config\OpenSwitch_ServerName.cfg`

- *OpenSwitch log file path* – is the location of the OpenSwitch log file; for example:

  `D:\Sybase\OpenSwitch-15_1\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 *servany.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_1\config\OpenSwitch_ServerName.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 87.

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

1 Shut down OpenSwitch using rp_shutdown (see rp_shutdown on page 224).

2 Restart OpenSwitch with the -E flag (see “Using command line options” on page 72).

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 72. 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 5, “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 *configuration file* – 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 See "[CONFIG]" on page 88 to configure "full pass-thru mode" using the *FULL_PASSTHRU* OpenSwitch configuration parameter.
- -h – prints usage message.
- -I *interfaces* – indicates that OpenSwitch should start using the specified *interfaces* file on UNIX or *sql.ini* file on Windows rather than the default *$SYBASE/interfaces* on UNIX or *%SYBASE%\ini\sql.ini* on 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 OPENSWITCH.

- -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 which are 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.

  You can also use the OpenSwitch Manager to set these flags.

Table 4-1: Trace flags

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*.

You can also use the OpenSwitch Manager to set these flags.

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

Table 4-2: Debug flags

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 rp_switch, 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 SRV_LANGUAGE event handler to *log_file*.
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.

Value	Description
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 rp_replay 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).

- -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.
- -V – disables the validation checks for all configuration options specified in the OpenSwitch configuration file.

  The -V option is provided when you start OpenSwitch server, as follows:

  ```
  ./OpenSwitch -c Absolute_path_to_the_configuration_file -V
  ```

  By default, these validations are turned on if -V is not specified.

  For example, *MUTUAL_AWARE* configuration parameter or option in the OpenSwitch configuration file is a boolean value whose valid values are 0 or 1. When -V option is not specified at server start time, which is the default, the *MUTUAL_AWARE* option will be validated for the specified value. If the value is neither 0 nor 1, an appropriate error message is displayed and OpenSwitch does not start.

  If the -V is specified, then *MUTUAL_AWARE* option will *not* be validated for its valid value of either 0 or 1.

- -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 155 for instructions.
- *RCMNAME* parameter in the RCM configuration file.

Requirements

See Chapter 4, “Using the Replication Coordination Module” in the *OpenSwitch Coordination Module Reference Manual*, 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 tool," in Chapter 3, "Configuring OpenSwitch" in 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 in *%OPENSWITCH%\config* inWindows and in *$OPENSWITCH/config* in UNIX.

4 Edit the following values in the file:

Parameter	Description
RCM_AUTOSTART	Instructs OpenSwitch whether to start the RCM. Enter: • 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.

Parameter	Description
RCM_RETRIES	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. Enter: • 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. This parameter is set to 1 when you choose Replication Coordination Module.
RCM_PATH	Enter the location where OpenSwitch should look for the RCM executable. If you do not enter a value, OpenSwitch runs the RCM located in *$OPENSWITCH/bin* on UNIX systems or in *%OPENSWITCH%\bin* on Windows systems; where *OPENSWITCH* is the installation directory. This parameter has a null value if you do not specify a path.
RCM_CFG_FILE	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.
RCM_LOG_FILE	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.
RCM_SECONDARY	Instructs OpenSwitch whether the RCM that is being started is a primary or a secondary RCM. Enter: • 0 – zero for a primary RCM. This is the default value. • 1 – for a secondary RCM.
RCM_TRC_FLAG	There is no command line equivalent for this option. Use this option to set trace flags for RCM when you start RCM from OpenSwitch. Enter: • 1 – to set trace flags for RCM. • 0 – to disable trace flags for RCM. This is the default. This is a dynamic option and you can set it with the rp_set registered procedure. See rp_set on page 218 for more details.

See “OpenSwitch sample configuration file” on page 80 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 155 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.

- UNIX:

```
[CONFIG]
SERVER_NAME       = rcm_osw
...
...
...
RCM_AUTOSTART     = 1
RCM_RETRIES       = 3
RCM_PATH          = /opt/sybase/OpenSwitch-15_1/bin/rcm
RCM_CFG_FILE      = /opt/sybase/OpenSwitch-15_1/config/rcm1.cfg
RCM_LOG_FILE      = /opt/sybase/OpenSwitch-15_1/logs/rcm.log
RCM_SECONDARY     = 0.........
```

- Windows:

```
[CONFIG]
SERVER_NAME       = rcm_osw
...
...
...
RCM_AUTOSTART     = 1
RCM_RETRIES       = 3
RCM_PATH          = C:\sybase\OpenSwitch-15_1\bin\rcm.exe
RCM_CFG_FILE      = C:\sybase\OpenSwitch-15_1\config\rcm1.cfg
RCM_LOG_FILE      = C:\sybase\OpenSwitch-15_1\logs\rcm.log
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
```

CHAPTER 5

Using the Configuration File

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

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* file in Windows or the *interfaces* file in UNIX. See Chapter 3, “Configuring OpenSwitch” in the *OpenSwitch Installation Guide* 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 a 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 any letter from “a” to “e” are channeled to POOL1.

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* in Windows or *$OPENSWITCH/config* in UNIX and copy the *sample.cfg* file by entering the following, where *OpenSwitch_ServerName* is the name of your server configuration file:

- On UNIX:

  ```
  cp sample.cfg OpenSwitch_ServerName.cfg
  ```

- On Windows:

  ```
  copy sample.cfg OpenSwitch_ServerName.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 87 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* on UNIX or *%OPENSWITCH%* on Windows and create a *logs* directory:

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 which has the *.cfg* file name extension; for example, *OpenSwitch_ServerName.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.

Note You can also use the OpenSwitch Manager management utility to change the configuration values of your OpenSwitch servers.

- [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 *NAME* options.

Format

```
[CONFIG]
NAME = VALUE
NAME = VALUE
NAME = VALUE
```

Parameters

NAME and *VALUE* – configuration options specific to OpenSwitch, which are entered in the format:

```
NAME=VALUE
```

where:

- *NAME* – is the option being set
- *VALUE* – is the option's setting

Table 5-1 on page 89 describes the options you can set (NAME), and the setting available for each option's VALUE.

The options shown in Table 5-1 are listed alphabetically for easy reference. This is not the order in which the options appear in the configuration file.

The value description also includes information on whether the option is static or dynamic:

- Dynamic option – takes effect immediately when you set it and it affects all future connections but does not affect existing connections. Dynamic options usually affect individual connections.
- Static option – you cannot change static options 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 this section.

In Table 5-1, the value description includes the command line option that you can set with this parameter. When you start OpenSwitch at the command line, you can enter the command line option to achieve the same results. See "Using command line options" on page 72.

Table 5-1: [CONFIG] NAME and VALUE options

Name	Value
ADMIN_PASSWORD	There is no command line equivalent for this option. Enter the administrative user password. If *USERNAME_PASSWORD_ENCRYPTED* 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 70. This is a dynamic option.
ADMIN_USER	There is no command line equivalent for this option. 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). If the *USERNAME_PASSWORD_ENCRYPTED* 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. This is a dynamic option.

Name	Value
API_CHECK	There is no command line equivalent for this option. This option indicates whether to enable or disable the validation of Server-Library arguments and state checking, and may be useful for debugging. Enter: • 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. This is a static option. **Warning!** Do not set *API_CHECK* to zero (FALSE) unless your application has been completely debugged with the default setting of 1 (TRUE). If *API_CHECK* 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.
BCP_LOGGED	There is no command line equivalent for this option. 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. Enter: • 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. This is a dynamic option.
CACHE_THREADS	There is no command line equivalent for this option. Enter an integer value that represents the maximum number of connection threads saved in the OpenSwitch server connection cache. This is a dynamic option.

Name	Value
CANCEL_QUERY_RESP_TIMEOUT	There is no command line equivalent for this option. Enter: • 1 – to cancel the query when the timeout period you set for a query with *RESPONSE_TIMEOUT* expires. • 0 – to allow the query to continue even when the timeout period set in *RESPONSE_TIMEOUT* expires. This is the default. This is a dynamic option.
CHARSET	Sets the -a command line option. Specifies the default character set used by OpenSwitch. This character set is used in communications with client connections. This is a static option.
CMON	There is no command line equivalent for this option. Enter: • 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 must also be set for OpenSwitch to allow mutually-aware support (MAS). *CMON* is set to 1 (one) by 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. This is a static option.
CMON_FAIL_ACTION	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL **Note** OpenSwitch currently supports only the DEFAULT action for *CMON_FAIL_ACTION*. This parameter is read-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 Table 5-1 on page 89 for information about the *CMON* parameter. See “CMON_FAIL_ACTION” on page 147and “User-specified actions” on page 141 for more information about this parameter and its actions.

Name	Value
CMON_PASSWORD	There is no command line equivalent for this option. Enter the password for *CMON_USER*. The default is (empty string). If the *USERNAME_PASSWORD_ENCRYPTED* 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 70. This is a static option.
CMON_RESPONSE_TIMEOUT	There is no command line equivalent for this option. This option allows the connection nonitoring thread to detect if Adaptive Server does not respond within the time you set. Enter an integer representing seconds. The default is 0 (zero). This value should be at least four times the values of *CMON_WAITFOR_DELAY*. You must set the *CMON_WAITFOR_DELAY* option when you use *CMON_RESPONSE_TIMEOUT*. This is a dynamic option.
CMON_USER	There is no command line equivalent for this option. 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. For *CMON_WAITFOR_DELAY* to work properly, this user should not be an administrative user on the remote data server. If the *CMON_USER* 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. If *USERNAME_PASSWORD_ENCRYPTED* 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. This is a static option.
CMON_WAITFOR_DELAY	There is no command line equivalent for this option. 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. Enter an integer representing seconds. The default is 3600 (1 hour). Set the *CMON_WAITFOR_DELAY* to a lesser value if you want the shutdown to occur more quickly. Use this when you want the graceful shutdown of an Adaptive Server rather than a shutdown with no wait. You must set the *CMON_WAITFOR_DELAY* option when you use *CMON_RESPONSE_TIMEOUT*. This is a dynamic option.

Name	Value
CMP_FAIL_ACTION **Note** Required only when `MUTUAL_AWARE=1`.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL 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. See “CMP_FAIL_ACTION” on page 148and “User-specified actions” on page 141 for more information about this parameter and its actions.
CON_TRACE	Sets the -C command line option. Set connection-level tracing flags. See “Using command line options” on page 72 for the available settings. The output of CON_TRACE is directed to the file specified by the *DEBUG_FILE*. This is a dynamic option.
CONNECTIONS	Sets the -u command line option. Enter the maximum number of user or threads the server can handle. **Note** Set this parameter to a value smaller than the maximum number of file descriptors (see UNIX ulimit -Ha command) to prevent OpenSwitch from using more descriptors than are available. This is a static option.

Name	Value
COORD_MODE	There is no command line equivalent for this option. Determines how OpenSwitch behaves when a coordination module (CM or RCM) is used. Enter: • 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. **Note** To use the RCM, you must set *COORD_MODE* 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 *OpenSwitch Coordination Module Reference Manual* for details. • 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. This is a dynamic option.
COORD_PASSWORD	There is no command line equivalent for this option. Enter the password used by the coordination module (CM or RCM) logging in as *COORD_USER*. 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. If *USERNAME_PASSWORD_ENCRYPTED* 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 70. This is a dynamic option.

Name	Value
COORD_TIMEOUT	There is no command line equivalent for this option. 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 *OpenSwitch Coordination Module Reference Manual* for more information about using concurrent coordination modules. This is a static option. **Note** If you are using an RCM, *COORD_TIMEOUT* must be set to zero (0) for the RCM to start.
COORD_USER	There is no command line equivalent for this option. 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. If *USERNAME_PASSWORD_ENCRYPTED* 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 70. This is a dynamic option.
CTX_TRACE	Sets the -X command line option. Enter context-level tracing flags. See “Using command line options” on page 72 for the available settings. The output of the CTX_TRACE is directed to the file specified by DEBUG_FILE. This is a static option.
CURSOR_PREREAD	There is no command line equivalent for this option. Enter an integer value that represents the number of rows to be returned with a single fetch request for a cursor. This is a dynamic option.
CUSTOM_SCRIPT	There is no command line equivalent for this option. The path to the user-created script to invoke. This is a dynamic option. **Note** When `MUTUAL_AWARE=1`, the scripts for both OpenSwitch companions must perform the same action. When a server fails over and *SVR_FAIL_ACTION* is set to MANUAL or CUSTOM, only one of the companions executes the script that notifies the administrator or restarts the server. See “Invoking custom and manual scripts” on page 139 for details about which exit codes to use in custom scripts.

Name	Value
DEBUG	Sets the -t command line option. Enter OpenSwitch debugging flags. See “Using command line options” on page 72 for valid flags to set. This is a dynamic option.
DEBUG_FILE	Sets the -d command line option. Enter the path to and name of the file in which to place debugging output. This option is used only for the context CTX_TRACE and connection CON_TRACE tracing debugging output. This is a static option.
ECHO_LOG	Sets the -e command line option. Enter: • 1 – to have all messages that are sent to the log display simultaneously to *stderr* 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 *stderr*. This is a static option.
FREEZE_CFG_ON_FAIL	There is no command line equivalent for this option. Whether OpenSwitch locks all server and pool configurations (forbids all changes) when a network break is suspected between the companions during *CMP_FAIL_ACTION*. Enter: • 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. This is a dynamic option.
FULL_PASSTHRU	Sets the -f command line option. 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: • 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. This is a static option.

Name	Value
HAFAILOVER	There is no command line equivalent for this option. This is property is required to enable *HAFAILOVER*. Enter: • 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. This is a dynamic option.
INTERFACES	Sets the -I (uppercase "i") command line option. Enter an alternate location for the *sql.ini* (Windows) or *interfaces* (UNIX) file, rather than the default of *%SYBASE%\ini\sql.ini* (Windows) and *$SYBASE/interfaces* (UNIX). This is a static option.
KEEPALIVE	There is no command line equivalent for this option. Enter: • 1 – to turn *KEEPALIVE* on. • 0 – to turn *KEEPALIVE* off. This is the default. See the Open Client documentation for CS_CON_KEEPALIVE for more information. This is a dynamic option.
LOG_FILE	Sets the -l command line option. Enter the file name in which to save all messages to *log_file* rather than the default of *OpenSwitch.log*. This is a static option.
LOGIN_TIMEOUT	There is no command line equivalent for this option. Enter the login timeout (CS_LOGIN_TIMEOUT property). The default is 60 seconds. This is a static option.
MANUAL_SCRIPT	There is no command line equivalent for this option. The path to the user-created manual script. This is a dynamic option. **Note** When you set `MUTUAL_AWARE=1`, the scripts for both OpenSwitch companions must perform the same action. 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 "Invoking custom and manual scripts" on page 139 for details about which exit codes to use in manual scripts.

Name	Value
MAX_LOG_MSG_SIZE	There is no command line equivalent for this option. Set this property when messages are bigger than 2048 characters. By default, OpenSwitch can log messages up to 2048 characters. This is a dynamic option.
MAX_LOGSIZE	There is no command line equivalent for this option. Enter the maximum size of the *log_file* (the default is 4194304). If *log_file* exceeds the size you set, OpenSwitch moves the current contents of *log_file* to a file named *currentfilename_old* and truncates the current log to 0 bytes. This is a static option.
MAX_PACKETSIZE	There is no command line equivalent for this option. Enter the maximum size of a TDS packet. The default is 2048. Used to tune throughput across the network. This is a static option. **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: `The packet size (2048) specified at login time is illegal. Legal values are between 512 and 512.` See "Connection refused" on page 30 for corrective actions.
MSGQ_SIZE	There is no command line equivalent for this option. Enter the deferred event queue size (SRV_S_DEFQUEUESIZE) for the context of OpenSwitch.The default is 2048. See the *Open Server Server-Library/C Reference Manual* for more information. This is a static option.
MUTUAL_AWARE	Specifies whether to use mutually-aware OpenSwitch servers. Enter: • 1 for a mutually-aware OpenSwitch. • 0 for a non-mutually-aware OpenSwitch, which is the default.
MUTUAL_CLUSTER **Note** Required only when `MUTUAL_AWARE=1`.	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 *CFG_STORAGE* servers (see the [SERVER] section). When this parameter is not set, it defaults to "CLUSTER1".

Name	Value
NET_FAIL_ACTION **Note** Required only when `MUTUAL_AWARE=1`. However, this parameter is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with *CMON* set to 1.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL This parameter is used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts. See "NET_FAIL_ACTION" on page 149 and "User-specified actions" on page 141 for more information about this parameter and its actions.
NOWAIT_ON_LOCKED	There is no command line equivalent for this option. 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. Enter: • 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 rp_switch 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. This is a dynamic option.
OPTIMIZE_TEXT	There is no command line equivalent for this option. Enter: • 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). This is a dynamic option.

Name	Value
PING_BINARY	There is no command line equivalent for this option. 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. This is a dynamic option.
PING_RETRIES	There is no command line equivalent for this option. 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. The default is “1”. This is a dynamic option.

Name	Value
PING_THREAD	There is no command line equivalent for this option. Enter: Valid values are 0 (zero) or 1. The default is 1. **Note** You can use *PING_THREAD* even if you do not enable mutually-aware support. • When set to 1, *PING_THREAD* detects the failure when an Adaptive Server host or network stops running. In a mutually-aware environment, the primary companion OpenSwitch checks to see if the entire network or only the Adaptive Server host network has failed. If the Adaptive Server host network has failed, the primary companion OpenSwitch is notified immediately and the action set for *SVR_FAIL_ACTION* is invoked. • When set to zero (0), *PING_THREAD* is not available to detect the network failure of an Adaptive Server proactively on behalf of OpenSwitch. However, OpenSwitch can still ping Adaptive Server and OpenSwitch receives a notification of a network failure of Adaptive Server. OpenSwitch receives the notification only after 8 minutes, which is the default TCP/IP *tcp_ip_abort_interval* configuration parameter. For mutually-aware support enabled OpenSwitch environments, you must use *PING_BINARY*, *PING_RETRIES*, and *PING_WAIT* in the configuration file, even if you set *PING_THREAD* to zero (0). OpenSwitch uses these parameters while it pings the companion OpenSwitch host and Adaptive Server hosts that store mutually-aware configuration information. These are Adaptive Server hosts which have *CFG_STORAGE* set to 1 in OpenSwitch configuration file. This is a static option. **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 *tcp_ip_abort_interval* 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. The length of strings holding the host name of a machine is limited to 30 characters within the OpenSwitch process. If a host name exceeds this limit, only the first 30 characters is used and may lead to incorrect results or failure during execution.

Name	Value
PING_WAIT	There is no command line equivalent for this option. The number of seconds that the ping command should wait before returning a failure. This parameter is used in conjunction with *PING_RETRIES*, on platforms where the ping command blocks instead of returning right away. The default is 10 (seconds). This is a dynamic option.
PRIMARY_COMPANION **Note** Required only when `MUTUAL_AWARE=1`.	Enter: • 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.
RCM_AUTOSTART	Instruct OpenSwitch whether to start the replication coordination module (RCM). Enter: • 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. This is a dynamic option.
RCM_CFG_FILE	Enter the path where the RCM configuration file is located. This parameter has a null value if you do not specify a path. This is a static option.
RCM_LOG_FILE	Enter the path where the RCM log file should be created. This parameter has a NULL value if you do not specify a path. This is a static option.
RCM_PATH	Enter the path where OpenSwitch should look for the RCM executable. If you do not enter this path, and are using an RCM, OpenSwitch runs the RCM located in *$OPENSWITCH/bin* on UNIX systems or in *%OPENSWITCH%\bin* on Windows systems; where *OPENSWITCH* is the installation directory. This parameter has a NULL value if you do not specify a path. This is a static option.
RCM_RETRIES	Enter how many times OpenSwitch should 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. • 0 – OpenSwitch does not attempt to restart the RCM. • Any numeric value – enter the number of times OpenSwitch should attempt to start the RCM. This is a static option.

Name	Value
RCM_SECONDARY	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). This is a dynamic option.
RCM_TRC_FLAG	There is no command line equivalent for this option. Use this option to set trace flags for RCM when you start RCM from OpenSwitch. Enter: • 1 – to set trace flags for RCM. • 0 – to disable trace flags for RCM. This is the default. This is a dynamic option and you can set it with the rp_set registered procedure. See rp_set on page 218 for more details.
RESPONSE_TIMEOUT	There is no command line equivalent for this option. Enter the timeout period for a response to a command (CS_TIMOUT property). The default is 60 seconds. This is a static option.
RMON	**Note** You must set this option for attributes under the [LIMIT_RESOURCE] section to take effect. Sets the -r command line option. Enter: • 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. This is a static option.
RMON_INTERVAL	There is no command line equivalent for this option. Set the frequency, in seconds, in which the resource monitoring thread awakens to check for offending connections.The default is 60 seconds. This is a static option.
RPC_SETFMT	There is no command line equivalent for this option. Enter: • 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. This is a dynamic option.

Name	Value
SEC_PRINCIPAL	Sets the -U command line option. 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. This is a static option.
SERVER_NAME	Sets the -n command line option. Enter the name in the *interfaces* file on UNIX and the *sql.ini* file on Windows by which the OpenSwitch is to be referred. This is a static option.
SHOW_CONNECT_ERROR	There is no command line equivalent for this option. Enter: • 1 – to have all ct_connect failure messages sent to the OpenSwitch log file. • 0 – the default. To not send ct_connect failure messages. This is a dynamic option.
SHOW_SPID	There is no command line equivalent for this option. Enter: • 0 – to prevent rp_who from showing the corresponding *spid* of a client connection to Adaptive Server. • 1 – to allow rp_who to show the corresponding *spid* of a client connection to Adaptive Server. See rp_who on page 237 for more information on this command. A value of 1 introduces a small overhead to each new connection and should be set only when necessary. The default is 0. This is a static option.
SITE_PASSTHRU	There is no command line equivalent for this option. Determines whether connections from site handlers can be passed through OpenSwitch. Enter: • 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. This is a dynamic option.
SSLIDENTITY_FILE	There is no command line equivalent for this option. Use this option to specify the absolute path to the file containing the digital certificate and the associated private key. This is a static option.

Name	Value
SSLIDENTITY_PASSWORD	There is no command line equivalent for this option. Use this option to decrypt the private key in the certificate file By default, OpenSwitch does not encrypt the password you specify with *SSLIDENTITY_PASSWORD* and you can read the password as it displays as clear text in the OpenSwitch configuration file. If you want to encrypt the password in the configuration file, see "Using encrypted user names and passwords" on page 70. This is a static option. See the *Open Client and Open Server Configuration Guide* for more information on client-side SSL.
SQL_WRAP	There is no command line equivalent for this option. 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. This is a dynamic option.
SRV_TRACE	Sets the -s command line option. Enter server-level tracing flags. The output is placed into the location of LOG_FILE. See "Using command line options" on page 72 for available settings. This is a static option.
STACKSIZE	Sets the -S command line option. Enter the stack size for each thread. The default is 40960. **Note** On 64-bit platforms, double this value to prevent a stack overflow. This is a static option.
SUPPRESS_CHARSET	There is no command line equivalent for this option. Enter: • 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 *stderr* if ECHO_LOG=1) when a client connects to OpenSwitch with a character set different from the one defined in CHARSET. This is a dynamic option.

Name	Value
SUPPRESS_DBCTX	There is no command line equivalent for this option. Enter: • 0 – to allow the database context change error message (5701). • 1 – suppresses the database context change error message (5701). The default is 1. This is a dynamic option.
SUPPRESS_LANG	There is no command line equivalent for this option. • 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. This is a dynamic option.
SVR_FAIL_ACTION **Note** Required only when `MUTUAL_AWARE=1`.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL 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. See “SVR_FAIL_ACTION” on page 150 for more information. See “User-specified actions” on page 141 for details about the available actions.

Name	Value
SWITCH_AT_LOGIN_TIMEOUT	There is no command line equivalent for this option. 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. **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. Enter: • 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. This is a dynamic option.
TCP_NODELAY	There is no command line equivalent for this option. Enter: • 1 – to turn TCP_NODELAY on. • 0 – to turn TCP_NODELAY off. This is the default. See the Open Client documentation for CS_CON_TCP_NODELAY for more information. This is a dynamic option.
TEXTSIZE	There is no command line equivalent for this option. 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. This is a dynamic option.
TRUNCATE_LOG	Sets the -T command line option. Enter: • 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. This is a static option.

Name	Value
UPDATE_CFG	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 `UPDATE_CFG=1`.

USE_AND_TO_POOL_ATTRIB	There is no command line equivalent for this option.
	Indicates whether all or only one of the connection attributes in the [POOL] section are to be satisfied by a client.
	Enter:
	• 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.
	For example:
	`[CONFIG]`
	`USE_AND_TO_POOL_ATTRIB = 1`
	`[POOl=POOL1:MODE=CHAINED,CACHE=0]`
	`servers:`
	`ASE1`
	`connections:`
	`username: john`
	`appname: isql`
	In this example, if USE_AND_TO_POOL_ATTRIB is set to 1, only clients with the user name of “john” and the application name of “isql” are allowed to use POOL1.
	If USE_AND_TO_POOL_ATTRIB is set to 0, any client with either the user name of “john” or the application name “isql” is allowed to use POOL1.
	This is a dynamic option.

Name	Value

USE_AND_TO_RMON_ATTRIB	There is no command line equivalent for this option.
	Indicates whether the connection attributes in the [LIMIT_RESOURCE] section can use "AND" or "OR."
	Enter:
	• 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.
	For example:
	`[CONFIG]`
	`RMON= 1`
	`USE_AND_TO_RMON_ATTRIB=1`
	
	`[LIMIT_RESOURCE:ACTION=KILL,BUSY=60]`
	`username: john`
	`appname: isql`
	`hostname: machine`
	`type:client`
	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.
	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.
	This is a dynamic option.

USE_DONEINPROCS	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).
USERNAME_PASSWORD_ENCRYPTED	There is no command line equivalent for this option. Enter: • 1– to indicate user name and password encryption. OpenSwitch expects *ADMIN_USER*, *ADMIN_PASSWORD*, *COORD_USER*, *COORD_PASSWORD*, *CMON_USER*, and *CMON_PASSWORD* to be encrypted. • 0 – to specify no user name and password encryption. OpenSwitch expects clear text for *ADMIN_USER*, *ADMIN_PASSWORD*, *COORD_USER*, *COORD_PASSWORD*, *CMON_USER*, and *CMON_PASSWORD*. This is a static option.

Example

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

- UNIX:

```
[CONFIG]
LOG_FILE = OpenSwitch.log
...
COORD_MODE         = ALWAYS
RCM_AUTOSTART      = 1
RCM_RETRIES        = 3
RCM_PATH           = /opt/sybase/OpenSwitch-15_1/bin/rcm
RCM_CFG_FILE       = /opt/sybase/OpenSwitch-15_1/config/rcm1.cfg
RCM_LOG_FILE       = /opt/sybase/OpenSwitch-15_1/logs/rcm.log
RCM_SECONDARY      = 0
...
...
...
```

- Windows:

```
[CONFIG]
LOG_FILE = OpenSwitch.log
...
COORD_MODE         = ALWAYS
RCM_AUTOSTART      = 1
RCM_RETRIES        = 3
RCM_PATH           = C:\sybase\OpenSwitch-15_1\bin\rcm.exe
RCM_CFG_FILE       = C:\sybase\OpenSwitch-15_1\config\rcm1.cfg
RCM_LOG_FILE       = C:\sybase\OpenSwitch-15_1\logs\rcm.log
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 5-2 on page 111 shows valid input for these parameters, which are entered in the format:

  ```
  OPTION=VALUE
  ```

Table 5-2: [SERVER] OPTION and VALUE settings

OPTION	VALUE

CFG_STORAGE	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.
	• 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.
	In a mutually-aware setup, you must include this parameter under the two Adaptive Server entries; for example:
	`[SERVER=ASE1]`
	`STATUS=UP`
	`CFG_STORAGE=1`
	
	`[SERVER=ASE2]`
	`STATUS=UP`
	`CFG_STORAGE=1`
	See Chapter 6, “Using Mutually-aware OpenSwitch Servers,” for more information.

OPTION	VALUE
STATUS	Enter the status of the server, which is the state of the server within OpenSwitch: • 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. 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. **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.

TYPE	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.
	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:
	`[CONFIG]`
	`HAFAILOVER = 1`
	`CMON_USER = cmon_user`
	`CMON_PASSWORD = cmon_password`
	
	`[SERVER = nlatke_ASE1]`
	`STATUS = UP`
	`TYPE = HA`

- *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.

  This value supersedes the value you set for *CMON_USER* in the [CONFIG] section. If you do not set a value here, OpenSwitch uses the value you set for *CMON_USER* in the [CONFIG] section.

  Also, if the *CMON_USER* 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.

If the *USERNAME_PASSWORD_ENCRYPTED* 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.

- *CMON_PASSWORD* – the password for the CMON_USER. The default is an empty string.

  This value supersedes the value you set for *CMON_PASSWORD* in the [CONFIG] section. If you do not set a value here, OpenSwitch uses the value you set for *CMON_PASSWORD* in the [CONFIG] section.

Examples

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.

```
[SERVER=ASE1]
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 5-3 shows valid input for these parameters, which are entered in the format `OPTION = VALUE`.

Table 5-3: [POOL] OPTION and VALUE settings

OPTION	VALUE
MODE	The operational mode of the pool: • 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.
STATUS	The status of the pool: • 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 *NOWAIT_ON_LOCKED* is set to 1, the clients immediately receive a descriptive error message from OpenSwitch. 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.
CACHE	The number of seconds that connections are cached following a user disconnect. See "Using connection caching" on page 33. The value is an integer in seconds.

- [servers:
  server_name=status
  server_name=status]

  "servers" is an optional [POOL] subsection that lists the servers in the pool, and optionally the server's status.

- *server_name* – the remote server defined in the *$SYBASE/interfaces* file on UNIX and *%SYBASE%\ini\sql.ini* on Windows.

  Note During OpenSwitch start-up, no validation is done to verify whether or not these servers exists.

- *status* – (optional) enter the status of the server you specified with *server_name* 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 *status* 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:

  - 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: • 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: • 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

[COMPANION=*companion_name*]
admin_user=ADMIN_USER
admin_password=ADMIN_PASSWORD

Parameters

- *companion_name* – the *SERVER_NAME* 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.
- *admin_user* – the *ADMIN_USER* from the companion OpenSwitch configuration file. This is a dynamic parameter, which means that you can change the value using rp_cfg while OpenSwitch is running, and the changes take effect immediately.
- *admin_password* – the *ADMIN_PASSWORD* from the companion OpenSwitch configuration file. This is a dynamic parameter, which means that you can change the value using rp_cfg while OpenSwitch is running, and the changes take effect immediately

Examples

```
[COMPANION=OSW2]
     admin_user=sa
     admin_password=sa
```

CHAPTER 6

Using Mutually-aware OpenSwitch Servers

This chapter describes the support in OpenSwitch 15.1 and later for mutually-aware OpenSwitch servers, and how to implement this functionality.

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.

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 121.

Concurrent coordination modules support

In OpenSwitch 15.1, a mutually-aware OpenSwitch server supports concurrent coordination modules.

A coordination module connects to an OpenSwitch server to control client logins and failover patterns. You can customize OpenSwitch to fit your requirements, and you can run multiple CMs against multiple OpenSwitch servers to create a redundancy environment so that no single OpenSwitch is a point of failure.

When multiple CMs connect to one OpenSwitch, these activities, which are transparent to the end user, take place:

1 Each CM registers its unique name with OpenSwitch using the Client-Library *CS_APPNAME* parameter. The unique name is generated by combining the host name and the process ID.

2 When the OpenSwitch server accepts a CM connection, it assigns the CM a unique ID number, and sends that number back to the CM as a message before the connection event is completed. OpenSwitch maintains an internal list of inactive CMs that are currently available.

3 If a CM becomes unresponsive for the period of time specified by the *COORD_TIMEOUT* configuration parameter, OpenSwitch retrieves the next CM ID number from the internal list of inactive CMs. All future notifications include the new ID as part of the notification.

For more information about using concurrent coordination modules, see Chapter 2, “Using Coordination Modules” in the *OpenSwitch Coordination Module Reference Guide*.

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 configuration GUI tool. You can also configure OpenSwitch after installation with the configuration GUI tool or manually by editing the OpenSwitch configuration file.

You must also specify the action OpenSwitch should take when certain failure types occur. See "Failure types" on page 140.

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:
  - *CMON_FAIL_ACTION*
  - *CMP_FAIL_ACTION*
  - *COORD_MODE*
  - *COORD_TIMEOUT*
  - *FREEZE_CFG_ON_FAIL*
  - *MUTUAL_AWARE*
  - *MUTUAL_CLUSTER*
  - *NET_FAIL_ACTION*
  - *SVR_FAIL_ACTION*
  - *UPDATE_CFG*
  - *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 in Windows, *interfaces* file in 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   hostname   4405
query   tcp   ether   hostname   4406

OSW2
query   tcp   ether   hostname   4406
query   tcp   ether   hostname   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* file in Windows or the *interfaces* file in UNIX.

- 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* file on Windows or *interfaces* file on UNIX 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.
- You can use concurrent coordination modules in a mutually-aware configuration in OpenSwitch 15.1 and later. See "Using concurrent coordination modules," in Chapter 2, "Using Coordination Modules" in the *OpenSwitch Coordination Module Reference Manual*.
- If you are using a CM or RCM in OpenSwitch 15.1 and later, you can set *SVR_FAIL_ACTION* to CUSTOM, MANUAL, CUSTOM_MANUAL, or DEFAULT.

  If you using OpenSwitch 15.0 and earlier, 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.

  Note The length of strings holding the host name of a machine is limited to 30 characters within the OpenSwitch process. If a host name exceeds this limit, only the first 30 characters is used and may lead to incorrect results or failure during execution.

- Start the primary OpenSwitch first, then start the secondary (companion) OpenSwitch. See Chapter 4, "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* in Windows or *interfaces* in 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 files is saved in the *SYBASE* root directory.

- *OpenSwitch_ServerName.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 6-1 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 6-1: Mutually-aware configuration parameters

Section	Parameter	Description
[CONFIG]	*CMON_FAIL_ACTION* **Note** Required when `MUTUAL_AWARE=1`. However, this parameter is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with *CMON* set to 1.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL **Note** OpenSwitch currently supports only the DEFAULT action for *CMON_FAIL_ACTION*. The selected action is invoked 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 Table 5-1 on page 89 for information about the *CMON* parameter. See “CMON_FAIL_ACTION” on page 147 for more information about this parameter. See “User-specified actions” on page 141 for details about each action.

Section	Parameter	Description
[CONFIG]	*CMP_FAIL_ACTION* **Note** Required only when `MUTUAL_AWARE=1`.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL 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. See "CMP_FAIL_ACTION" on page 148 for more information about this parameter. See "User-specified actions" on page 141 for details about each action.
[CONFIG]	*CUSTOM_SCRIPT*	The path to the user-created script to invoke. **Note** When `MUTUAL_AWARE=1`, the scripts for both OpenSwitch companions must perform the same action. When a server fails over and *SVR_FAIL_ACTION* is set to to MANUAL or CUSTOM, only one of the companions executes the script that notifies the administrator or restarts the server. See "User-specified actions" on page 141 for details about which exit codes to use in custom scripts.
[CONFIG]	*FREEZE_CFG_ON_FAIL*	Whether OpenSwitch locks all server and pool configurations (forbids all changes) when a network break is suspected between the companions during *CMP_FAIL_ACTION*. Enter: • 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]	*MANUAL_SCRIPT*	The path to the user-created manual script. See "User-specified actions" on page 141 for details about which exit codes to use in manual scripts.

Section	Parameter	Description
[CONFIG]	*MUTUAL_AWARE*	Specifies whether to use mutually-aware OpenSwitch servers. Enter: • 1 for a mutually-aware OpenSwitch. • 0 for a non-mutually-aware OpenSwitch, which is the default.
[CONFIG]	*MUTUAL_CLUSTER* **Note** Required only when `MUTUAL_AWARE=1`.	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 *CFG_STORAGE* servers (see "[SERVER]" on page 110). When this parameter is not set, it defaults to "CLUSTER1".
[CONFIG]	*NET_FAIL_ACTION* **Note** Required only when `MUTUAL_AWARE=1`.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL This parameter is used when the local OpenSwitch is experiencing a network outage and cannot communicate with the Adaptive Servers or companion OpenSwitch hosts. See "NET_FAIL_ACTION" on page 149 for more information about this parameter. See "User-specified actions" on page 141 for details about each action.
[CONFIG]	*PING_BINARY*	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.
[CONFIG]	*PING_RETRIES*	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. The default is "1".

Section	Parameter	Description
[CONFIG]	*PING_THREAD*	Valid values are 0 (zero) or 1. The default is 1. When set to 1, PING_THREAD detects the failure when an Adaptive Server host or network stops running. In a mutually-aware environment, 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 *SVR_FAIL_ACTION* is invoked. 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 *tcp_ip_abort_interval* configuration parameter. **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 *tcp_ip_abort_interval* 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. The length of strings holding the host name of a machine is limited to 30 characters within the OpenSwitch process. If a host name exceeds this limit, only the first 30 characters is used and may lead to incorrect results or failure during execution.
[CONFIG]	*PING_WAIT*	The number of seconds that the ping command should wait before returning a failure. This parameter is used in conjunction with *PING_RETRIES*, on platforms where the ping command blocks instead of returning right away. The default is 10 (seconds).
[CONFIG]	*PRIMARY_COMPANION* **Note** Required only when `MUTUAL_AWARE=1`.	Enter: • 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]	*SVR_FAIL_ACTION* **Note** Required only when `MUTUAL_AWARE=1`.	Enter one of these actions: • DEFAULT • CUSTOM • MANUAL • CUSTOM_MANUAL 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. See "SVR_FAIL_ACTION" on page 150 for more information. See "User-specified actions" on page 141 for details about the available actions.
[CONFIG]	*UPDATE_CFG*	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 `UPDATE_CFG=1`.

[SERVER]	*CFG_STORAGE*	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:
		• 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.
		In a mutually-aware setup, you must include this parameter under the two Adaptive Server entries; for example:
		`[SERVER=ASE1]`
		`STATUS=UP`
		`CFG_STORAGE=1`
		
		`[SERVER=ASE2]`
		`STATUS=UP`
		`CFG_STORAGE=1`
		In the example above, the most recent OpenSwitch configuration is stored in the syso_<cluster> 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.

Section	Parameter	Description

[COMPANION]	*companion_name*
	admin_user
	admin_password
		Name of the OpenSwitch companion, and the administrator user name and password used to make a connection. For example:
		`[COMPANION=OSW2]`
		`admin_user=sa`
		`admin_password=sa`
		These should be the same as the values for *SERVER_NAME*, *ADMIN_USER* and *ADMIN_PASSWORD* in the companion OpenSwitch configuration file.
		companion_name is a static parameter, while *admin_user* and *admin_password* are dynamic parameters. This means that after an OpenSwitch server starts running, you can change the companion's *admin_user* and *admin_password* using rp_cfg and the changes take effect immediately. However, if you change *companion_name*, you must restart the OpenSwitch server for the change to take effect.
		Note If *USERNAME_PASSWORD_ENCRYPTED* is set to 1, *admin_user* and *admin_password* should contain the encrypted string (as should *ADMIN_USER* and *ADMIN_PASSWORD* in the OpenSwitch configuration file). Use the -E or -p command line options for encryption. See "Using encrypted user names and passwords" on page 70.

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 6-2 on page 133 lists the columns in the mutually-aware configuration table created in the Adaptive Servers:

Table 6-2: 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: • 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 121.

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 65 or "Starting and stopping OpenSwitch on Windows" on page 66 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 139 for more information about **_FAIL_ACTION* functionality.

Failback

Failback is a manual process that requires careful planning. If you use an RCM and it is down, in order to fail back, you may need to manually restart the RCM, and manually reverse 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 78 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.

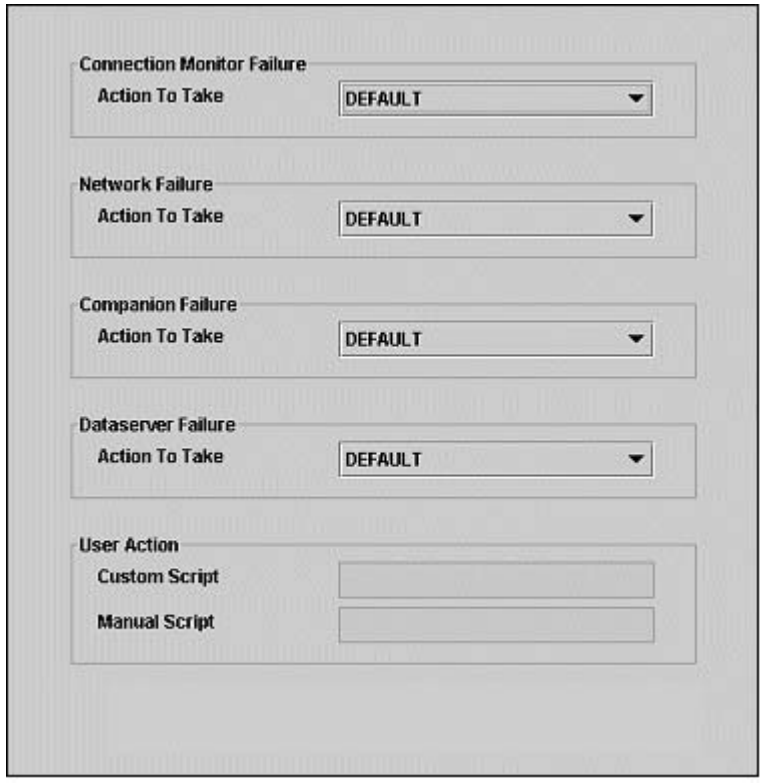
  Note *CMON_FAIL_ACTION* is not specific to only mutually-aware OpenSwitch servers; it applies to all OpenSwitch servers with *CMON* set to 1.

- 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 6-1 on page 141.

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 6-1: Mutually-aware failure types and actions



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 147, "CMP_FAIL_ACTION" on page 148, "NET_FAIL_ACTION" on page 149, and "SVR_FAIL_ACTION" on page 150.

DEFAULT

Invokes the default OpenSwitch behavior for the failure type. The activity that occurs depends on the failure type. See "Failure types" on page 140.

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 148, “SVR_FAIL_ACTION” on page 150, and “NET_FAIL_ACTION” on page 149 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 6-3 on page 146 lists the reason codes associated with each failure type.

Table 6-3: Custom and manual script reason codes

Code	Failure type	Description
1000	*CMON_FAIL_ACTION*	The Adaptive Server cannot be connected to because it is not running.
1001	*CMON_FAIL_ACTION*	The number of connections to the Adaptive Server has exceeded the maximum user connections limit.
1004	*SVR_FAIL_ACTION*	A primary Adaptive Server failed, and a restart or failover is necessary.
1005	*NET_FAIL_ACTION*	The network of the local OpenSwitch host is disabled.
1006	*CMP_FAIL_ACTION*	The companion OpenSwitch stops unexpectedly.
2000	*CMON_FAIL_ACTION*	The user name or password for the CMON user is invalid on this Adaptive Server.
3000	*CMON_FAIL_ACTION*	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 147, “CMP_FAIL_ACTION” on page 148, “NET_FAIL_ACTION” on page 149, and “SVR_FAIL_ACTION” on page 150 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 88 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 6-3 on page 146 for descriptions of the reason codes. OpenSwitch is suspended and waits indefinitely until the system administrator executes rp_go.

Note OpenSwitch currently supports only the DEFAULT action for *CMON_FAIL_ACTION*.

See "User-specified actions" on page 141 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

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.

Actions and reason codes

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 88 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 141 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 141 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

SVR_FAIL_ACTION is used when an Adaptive Server does not respond in a timely manner, or when the Adaptive Server host cannot be pinged by either OpenSwitch server in a cluster.

SVR_FAIL_ACTION can be used if a CM is connected to the OpenSwitch.

For example, you have two OpenSwitch servers, OSW1 and OSW2. OSW1 detects a failure first, but if it does not have a CM connection, OSW1 cannot handle the failure. If OSW2 has a CM connection, it handles the failure. If OSW1 is a primary server, and OSW2 is a secondary server, and both detect the failure, OSW2 handles the failure only if it has a CM and OSW1 does not.

With OpenSwitch 15.1 and later, if you are using a CM or RCM, you can set *SVR_FAIL_ACTION* to CUSTOM, MANUAL, CUSTOM_MANUAL, or DEFAULT.

With OpenSwitch 15.0 and earlier, you cannot run custom or manual scripts for a server failure because the failover procedures for CMs and RCMs differ from each other, and 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, and:

- If *SVR_FAIL_ACTION* is set to either DEFAULT or MANUAL, the CM or RCM performs the normal failover.
- If *SVR_FAIL_ACTION* is set to CUSTOM and the script returns an exit code of either 0 or an invalid code, the CM or RCM performs the normal failover.
- If *SVR_FAIL_ACTION* is set to CUSTOM and the script returns an exit code of either 1 or 3, the CM or RCM performs the failover without pinging the Adaptive Server host.
- If *SVR_FAIL_ACTION* is set to CUSTOM and the script returns an exit code of 2, the CM or RCM does not perform any action and OpenSwitch terminates all the existing connections.

When there are no CM or RCM connections, OpenSwitch:

1 Marks the failed primary 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.

The status of the Adaptive Server is SUSPENDED when *SVR_FAIL_ACTION* is invoked and the status changes to UNSUSPENDED after *SVR_FAIL_ACTION* completes.

- 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 141 for additional details about these actions.

Exit codes for custom scripts

Valid exit codes are:

- 0 – the script is successful and OpenSwitch should reconnect all existing clients to the same primary Adaptive Server. The script should return this exit code if it restarts the primary Adaptive Server successfully.

  OpenSwitch does not change the status of the primary Adaptive Server to DOWN and OpenSwitch continues to route future connections to the same primary Adaptive Server.

- 1 – the script is successful and OpenSwitch should fail over all existing clients to the secondary Adaptive Server. The script should return this exit code if it 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 OpenSwitch routes future connections to the next available Adaptive Server in the pool.

- 2 – the script is unsuccessful and OpenSwitch should terminate all existing client connections. The script should return this exit code if the script fails and OpenSwitch does not perform an automatic failover.

The primary Adaptive Server status changes to LOCKED. If the primary Adaptive Server does not restart, OpenSwitch blocks new client connections until the status changes to either UP or DOWN, or the client times out. Client connections reconnect to the primary or secondary Adaptive Server, depending on the actions of the administrator:

- If the primary Adaptive Server restarts and the administrator changes the status of the primary Adaptive Server to UP, client connections reconnect to the primary Adaptive Server.
- If the primary Adaptive Server does not restart but the administrator changes the status of the primary Adaptive Server to UP, client connections connect to the secondary Adaptive Server.
- If the primary Adaptive Server restarts but the administrator changes the status of the primary Adaptive Server to DOWN, client connections connect to the secondary Adaptive Server.
- If the primary Adaptive Server does not restart and the administrator changes the status of the primary Adaptive Server to DOWN, client connections connect to the secondary Adaptive Server.

- 3 – the script is unsuccessful and OpenSwitch should fail over all existing clients to the secondary Adaptive Server. This script should return this exit code if the script fails, 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 OpenSwitch routes all future connections to the next available Adaptive Server in the pool.

- Any exit code that is more than three (3) – perform the default operation. For example, setting the primary Adaptive Server status to DOWN.

Exit codes for manual scripts

Valid exit codes are:

- 0 – the script is successful and OpenSwitch should reconnect all existing clients to the same primary Adaptive Server. The script should return this exit code if it restarts the primary Adaptive Server successfully.

  OpenSwitch does not change the status of the primary Adaptive Server to DOWN and OpenSwitch continues to route future connections to the same primary Adaptive Server.

- Any exit code that is more than zero (0) – perform the default operation. For example, setting the primary Adaptive Server status to DOWN.

CHAPTER 7

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.

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_ServerName* is the name of the OpenSwitch server in the interfaces file being used by the Adaptive Server:

```
sp_addserver OpenSwitch_ServerName
```

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_ServerName...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.

Table of registered procedures

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.

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_cm_list	Displays a list of coordination modules connected to OpenSwitch.
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 or a specified Adaptive Server after a manual intervention has been requested and performed by OpenSwitch.

Registered procedure	Description
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_rematttrib	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.
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 *spid*. rp_showquery 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 rp_stop.
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.

Registered procedure	Description
rp_who	Displays information about the current set of user connections. Similar to sp_who.

rp_cancel

Description

Cancels processing of a client connection.

Syntax

rp_cancel [*pool_name*, *srv_name*, *spid*, *why*]

Parameters

pool_name

The name of the pool in which the connections should be canceled. Supplying only this argument causes all connections within *pool_name* to be canceled.

srv_name

Cancels connections to remote server *srv_name*. Supplying only this argument cancels all connections to *srv_name*.

spid

Cancels the connection identified within OpenSwitch by *spid*. If this parameter is specified while *pool_name* and *srv_name* are both NULL, OpenSwitch cancels the *spid* as specified by this parameter.

If either *pool_name* or *srv_name* are not NULL, those parameters are verified against the *actual pool_name* or *srv_name* of the specified *spid*. The connection for the specified *spid* is cancelled only if the supplied *pool_name* and *srv_name* exactly match the *actual _pool_name* and *srv_name* that the specified *spid* is connected to. Otherwise, no connection is cancelled.

why

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*)”.

Examples

Example 1 Cancels the OpenSwitch connection represented by spid 8.

```
1> rp_cancel NULL, NULL, 8
2> go
```

Example 2 Cancels all connections currently established to pool POOL_A.

```
1> rp_cancel "POOL_A", NULL, NULL
2> go
```

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.

```
1> rp_cancel "POOL_A", "SYB_SERV2", NULL, "Sorry! Your connection was
canceled."
2> go
```

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_ServerName.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_ServerName.cfg* file in */usr/sybase/config* on the same host as the OpenSwitch server.

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

On the Windows platform:

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

This command processes the *OpenSwitch_ServerName.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 7-1.

Table 7-1: Effects of rp_cfg on existing configuration settings

Section	Behavior
[CONFIG]	As new *name=value* pairs are processed within this section, the current value of *name* in OpenSwitch is replaced with *value*. Also, some variables, such as *RMON*, are meaningful only during start-up, so changing the value of these variables at runtime has no effect (for example, setting *RMON* 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 *admin_user* and *admin_password* 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 rp_cfg.

Section	Behavior
[POOL]	All existing pool information is replaced with the contents of the new configuration file.
[LIMIT_RESOURCE]	When the new configuration file is processed, all existing [LIMIT_RESOURCE] settings are cleared and replaced by the contents of this section.

- After verifying that the specified *config_file* exists, rp_cfg 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.
- rp_cfg 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 rp_cfg executes quickly and the window is almost negligible. If this causes concern, run rp_cfg when there is are fewer clients connecting to OpenSwitch.
- Do not use rp_cfg to reconfigure OpenSwitch; use another registered procedure call instead. For example, to change a [CONFIG] parameter, use rp_set (see rp_set on page 218); to add a server from to pool, use rp_pool_addserver (see rp_pool_addserver on page 179); to remove a server from a pool, use rp_pool_remserver (see rp_pool_remserver on page 194).
- Do not use rp_cfg 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_cm_list

Description

Displays all the Coordination Modules that are currently connected to the OpenSwitch server.

Note You can also use OpenSwitch Manager to view the coordination modules connected to the OpenSwitch server.

Syntax

rp_cm_list

Parameters

None.

Examples

Log in to OpenSwitch as an administrator and enter the following commands to list all the Coordination Modules:

```
1> rp_cm_list
2> go
```

Usage

- Used to check which CMs are connected to an OpenSwitch by displaying the CM’s name and its status when the concurrent CM feature is enabled.

- Used to check which CMs are connected to an OpenSwitch by displaying the CM’s name and its status when the pre-concurrent CM feature is enabled.

rp_debug

Description

Enables or disables OpenSwitch debugging messages.

Note You can also use the OpenSwitch Manager to set the debug options of your OpenSwitch servers. See “Editing OpenSwitch server properties” on page 51 for more information.

Syntax

rp_debug [*options*, [on|off]]

Parameters

options

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.

Table 7-2 shows the valid debugging options. These are identical to the options you can use with the -t flag at the command line.

Table 7-2: 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 rp_switch, 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 SRV_LANGUAGE event handler to *log_file*.
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 rp_replay 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 7-2 on page 166.

rp_dump

Description — Dumps connection state information.

Syntax — rp_dump [*@what*, *@sentolog*]

Parameters

@what

Specifies the type of connection state information:

- thread – dumps information about all user connection threads in OpenSwitch.
- mutex – dumps information about all mutexes that OpenSwitch owns.
- all – dumps information about all OpenSwitch threads and mutexes. This is the default if you do not specify the type of connection state information.

@sendtolog

Specifies whether to write the dump of the connection state information to the OpenSwitch log:

- 0 – OpenSwitch does not write the dump of the connection state to the OpenSwitch log. This is the default.
- 1 – OpenSwitch writes the dump of the connection state information to the OpenSwitch log.

Examples

```
1> rp_dump thread
2> go
```

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

- Use this procedure for debugging.
- Use rp_dump when you want more details about user connections.

See also — rp_who

rp_go

Description

Resumes OpenSwitch activities that were previously suspended by various failure points. See “Failure types” on page 140 for more information.

Syntax

rp_go [*srv_name*]

Parameters

srv_name

Provide the Adaptive Server name to resume server activities after the manual script for *SVR_FAIL_ACTION* executes successfully.

If you do not specify any Adaptive Server name, rp_go resumes all pool activities after these failure types are executed: *CMP_FAIL_ACTION* and *NET_FAIL_ACTION*.

Usage

- Issued during an Adaptive Server, companion OpenSwitch failure, or network failure after a MANUAL intervention has been requested and performed by the administrator. Failure types are *CMP_FAIL_ACTION*, *NET_FAIL_ACTION*, *SVR_FAIL_ACTION*.
- Returns all pools to their original states and returns the Adaptive Server to its original state if you specify the Adaptive Server with rp_go. This allows all pending clients applications that are running to log in.
- 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.

See also

“Invoking custom and manual scripts” on page 139.

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_remattrib              @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.

Note You can also use OpenSwitch Manager to terminate connections.

Syntax

rp_kill [*pool_name*, *srv_name*, *spid*]

Parameters

pool_name

The name of the pool for which the connections should be shut down. Supplying only this argument causes all connections within *pool_name* to be shut down.

srv_name

Shuts down connections to remote server *srv_name*. Supplying only this argument causes all connections to *srv_name* to be shut down.

spid

Shuts down the connection identified within OpenSwitch by *spid*. If this argument is specified, *pool_name* and *srv_name* are ignored.

Examples

Example 1 Shuts down the OpenSwitch connection represented by *spid* 8 (as identified by rp_who).

```
1> rp_kill NULL, NULL, 8
2> go
```

Example 2 Shuts down all connections currently established to pool POOL_A.

```
1> rp_kill "POOL_A", NULL, NULL
2> go
```

Example 3 Shuts down all connections currently established to pool POOL_A on server SYB_SERV2.

```
1> rp_kill "POOL_A", "SYB_SERV2", NULL
2> go
```

Usage

- If no arguments are supplied to rp_kill, all connections are killed within OpenSwitch. Use this procedure with caution.
- *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.
- Shutting down a connection causes it to be forcibly removed from OpenSwitch. No messages are delivered to the client.

Messages

- Indicates the number of OpenSwitch connections that were killed:

  ```
  rp_kill: Killed n spids.
  ```

- 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

rp_msg [*pool_name*], [*srv_name*], [*spid*], *msg*

Parameters

pool_name

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.

srv_name

Sends the message to connections currently established to *srv_name*.

spid

The OpenSwitch process ID of the client connection to receive the message. Connection spid numbers can be obtained using rp_who.

msg

The text of the message to be delivered. This message must be less than 255 characters in length.

Examples

```
1> rp_msg POOL1, ase2, 19, "TEST"
2> go
```

Results:

```
Successfully queued message to spid 19
(return status = 0)
```

Usage

- 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 rp_msg 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 rp_msg from a client connection that is passing through OpenSwitch and using Adaptive Server to invoke the rp_msg RPC in OpenSwitch. This causes the issuing connection to be locked in a transaction, which means that the connection cannot be switched.

Messages

- See the OpenSwitch error log (specified via the -l flag or *LOG_FILE* 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.

Note You can also use OpenSwitch Manager to set pool attributes for an OpenSwitch server.

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 7-3 shows the valid values.

Table 7-3: Attribute names for rp_pool_addattrib

Attribute	Description
username	*value* must match the user name of an incoming client connection.
appname	*value* must match the application name identified by the incoming client connection.
hostname	*value* 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 186.

- 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_remattrib

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.

Note You can also use OpenSwitch Manager to add servers to a pool.

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:

Table 7-4: Adaptive Server states

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:

Table 7-5: Server positions

Value	Description
HEAD	*rel_server* is ignored, and *server* is placed at the beginning of the list of servers in the pool
TAIL	*rel_server* is ignored, and *server* is placed at the end of the list of servers in the pool. This is the default position if no value is supplied for *position*.
BEFORE	*server* is added immediately before *rel_server* in the list of servers in the pool.
AFTER	*server* is added immediately after *rel_server* 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 33.

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.

Note You can also use OpenSwitch Manager to create pools.

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 7-6 describes the values for *position*.

Table 7-6: Position values for rp_pool_create

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

status

The initial status of the pool. Table 7-7 describes the values for *status*.

Table 7-7: 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 7-8 on page 187 describes the values for *mode*.

Table 7-8: 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 177.
- 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.

Note You can also use OpenSwitch Manager to remove pools.

Syntax

rp_pool_drop *pool_name*

Parameters

pool_name

Name of the pool to be dropped.

Examples

Drops the pool named “POOL2” from the pool list in the OpenSwitch server.

```
1> rp_pool_drop POOL2
2> go
```

Returns:

```
rp_pool_drop: Pool  POOL2 dropped
(return status = 0)
```

Usage

- 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 *config* file.
- You can use rp_pool_help to get a list of the current pool names.

Messages

- You did not supply a pool name for the pool to be dropped:

```
Procedure rp_pool_drop expects parameter @pool_name,
which was not supplied
```

- The pool with the supplied pool name does not exist:

```
rp_pool_drop: There is no such pool as 'pool_name'
```

Run rp_pool_status pool_name to check your pool list.

See also

rp_pool_create, rp_pool_status

rp_pool_help

Description

Displays information about pools.

Note You can also use OpenSwitch Manager to view 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 7-9 describes the values for *attrib*.

Table 7-9: Attribute names for rp_pool_remattrib

Attribute	Description
username	Value must match the user name of an incoming client connection.
appname	Value must match the application name identified by the incoming client connection.
hostname	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.

Note You can also use OpenSwitch Manager to remove servers from a pool.

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.

Note You can also use OpenSwitch Manager to view or set the status of servers.

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.

Note You can also use OpenSwitch Manager to change the pool status.

Syntax

rp_pool_status [*pool_name*, *status*]

Parameters

pool_name

Name of the pool to be displayed or changed.

status

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 7-10 describes the values for *status*.

Table 7-10: Status values for rp_pool_status

Status	Description
PRE_UP	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. **Warning!** Do not manually set a pool's status to PRE_UP.
UP	The pool is immediately available for use.
PRE_DOWN	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. **Warning!** Do not manually set a pool's status to PRE_DOWN.
DOWN	The pool is unavailable, and is not considered for use by any new client connections established to OpenSwitch.

Status	Description
PRE_LOCKED	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. **Warning!** Do not manually set a pool's status to PRE_LOCKED.
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 139 for more information. The pool blocks on all new connections until rp_go is issued. **Warning!** Do not manually set a pool's status to SUSPENDED.

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	rp_rcm_connect_primary
Parameters	None.
Usage	In a redundant RCM environment, issue rp_rcm_connect_primary on the secondary OpenSwitch after the primary OpenSwitch has gone down and been restarted.
See also	rp_rcm_list

rp_rcm_list

Description

Displays a list of RCMs known to the OpenSwitch on which rp_rcm_list is executed. The list displays the RCM name and whether the RCM is a primary or secondary RCM.

Note You can also use OpenSwitch Manager to view RCMs known to the OpenSwitch server.

Syntax

rp_rcm_list

Parameters

None.

Usage

- Used to check which RCMs are connected to an OpenSwitch.
- You can also use rp_rcm_list to generate the name of the RCM to shut down through OpenSwitch using rp_rcm_shutdown.

See also

rp_rcm_shutdown, rp_rcm_connect_primary, rp_rcm_startup

rp_rcm_shutdown

Description — Shuts down the named RCM through OpenSwitch.

Syntax — rp_rcm_shutdown *rcm_name*

Parameters — *rcm_name*

The name of the RCM specified for the *RCM_NAME* parameter in the RCM configuration file.

- If you do not specify *rcm_name*, this parameter's value defaults to "***OPENSWITCH***_rcm," where *OPENSWITCH* is another parameter set in the RCM configuration file.
- If *RCMNAME* is not set in the RCM configuration file, enter "***OPENSWITCH***_rcm" for *rcm_name* to shut down the RCM that was started by this OpenSwitch.

Examples — Shuts down the RCM named "osw_primary_rcm."

```
1> rp_rcm_shutdown osw_primary_rcm
2> go
```

Returns:

```
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)
```

Usage — Shuts down an RCM through OpenSwitch. The RCM does not have to have been started by OpenSwitch to be shut down using rp_rcm_shutdown.

After issuing rp_rcm_shutdown, execute rp_rcm_list and ps to verify that the RCM has shut down.

See also — rp_rcm_list, rp_rcm_startup

rp_rcm_startup

Description — Starts an RCM using the specified path and configuration file.

Syntax — rp_rcm_startup [*rcm_path*, *rcm_cfg*, *rcm_log*, *rcm_retries*, *rcm_redundant*]

Parameters

rcm_path
Path to the RCM binary.

- If not provided (NULL), the RCM binary is executed using the value of the *RCM_PATH* parameter set in the OpenSwitch configuration file. If *RCM_PATH* is not set, the RCM binary is executed from *$OPENSWITCH/bin/rcm* on UNIX and from *%OPENSWITCH%\bin\rcm* on Windows.
- When you provide rcm_path, the RCM executable is invoked from the specified location. You must have execution permission on the RCM binary.

rcm_cfg
Absolute path to the RCM configuration file. If not provided, the value specified by the *RCM_CFG_FILE* parameter in the OpenSwitch configuration file is used. You must have read access on this file.

rcm_log
Absolute path to the RCM log. If not provided, the value specified by the *RCM_LOG_FILE* parameter in the OpenSwitch configuration file is used.

rcm_retries
An integer value to specify the number of times OpenSwitch restarts the RCM if it fails. If NULL, the value specified by *RCM_RETRIES* parameter in the OpenSwitch configuration file is used.

rcm_redundant
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.

Examples — Starts the RCM with the default RCM binary, trying twice, with no redundancy.

```
1> rp_rcm_startup NULL, /scratch/15.0_bld2/OpenSwitch-15_1/config/rcm.cfg,
   /scratch/15.0_bld2/OpenSwitch-15_1/logs/rcm.log, 2, 0
2> go
```

Returns:

```
Msg 20107, Level 16, State 0:
Server 'monsoon_OSW':rp_rcm_startup:
```

```
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:

```
Messsage – test_osw: INFO: spid 10: thrd_sql_add:
@name should be unique, 'insert into temp1 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.

Note You can also use OpenSwitch Manager to view attribute/value pairs used by the resource governor.

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.

Note You can also use OpenSwitch Manager to view information about a server.

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.

Note You can also use OpenSwitch Manager to view and change the status of a server.

Syntax

rp_server_status [*server*, *status*]

Parameters

server

The name of a remote server as listed in the *interfaces* file on UNIX, or *sql.ini* 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 *status* is not supplied, the status of the specified server is displayed.

status

The disposition to which the server is to be changed. If *status* is supplied, but a server is not, then the current status of all servers is changed.

Table 7-11: Status values for rp_server_status

Status	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.

Status	Description
PRE_LOCKED	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. **Warning!** Do not manually set a server's status to PRE_LOCKED.
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.

Note You can also use OpenSwitch Manager to view and set the configuration parameters.

Syntax

rp_set [*parm_name*, *parm_value*]

Parameters

parm_name

Name of a parameter as listed in the configuration file.

parm_value

Value to which the parameter is to be set. The meaning of the value depends on which parameter is being set.

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 Sets the TEXTSIZE configuration parameter to NULL.

```
rp_set TEXTSIZE, -1
```

Example 4 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_KEEPALIVE              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.
- If you set the *parm_value* parameter to -1, rp_set sets the configuration parameter to null.
- 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 5, “Using the Configuration File.”

rp_set_srv

Description — Responds to *spid* waiting for coordination module response.

Syntax — rp_set_srv *spid*, *srv_name*

Parameters

spid
The OpenSwitch process identifier of the connection waiting for a response from a coordination module.

srv_name
The name of the remote server to which the connection, represented by *spid*, should be switched. Passing a server name of NULL or "" causes the *spid* to switch to the next available server according to the mode of the pool to which it belongs.

Examples

Example 1 Connects *spid* 8 to SYB_SERV3.

```
rp_set_srv 8, "SYB_SERV3"
```

Example 2 Connects *spid* 8 to the next available server in its pool.

```
rp_set_srv 8, ""
```

Usage

- 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 np_req_srv notification procedure call that is detected by the coordination module (see "np_req_srv" on page 242). The requesting connection suspends until the coordination module responds with a call to rp_set_srv, rp_switch, or rp_kill.
- rp_set_srv is intended for use by the coordination module to set the server for a *spid* through cm_set_srv, but an administrator can also manually set the server for a *spid* if the *spid* is at a "SERVER REQ" state. This can happen if *COORD_MODE* is set to ALWAYS and no coordination module is running.
- You can list the set of *spids* awaiting a response from the coordination module using the Open Server built-in registered procedure sp_ps. These *spids* display a *sleep_label* of "SERVER REQ".
- Calling rp_req_srv on a *spid* that is not actively waiting for a response from a coordination module has no effect.

See also — rp_kill, rp_switch

rp_showquery

Description

Displays query being executed by a *spid*. rp_showquery works only when OpenSwitch in not in FULL_PASSTHRU mode.

Syntax

rp_showquery *spid*

Parameters

spid

The OpenSwitch process identifier of the connection executing a query.

Examples

```
1> rp_showquery 8
2> go
```

Returns:

```
Statement
------------
sp_who
```

Usage

- Use rp_who to list valid *spids*.
- Output is displayed only for *spids* that are actively processing a query (the state field of rp_who shows BUSY), or executing registered procedures or SQL queries. Cursors are not currently supported.

Messages

- *spid x* is not a valid *spid* as listed in rp_who:

```
rp_showquery: spid #x is invalid.
```

See also

rp_who

rp_shutdown

Description	Shuts down the OpenSwitch server.
Syntax	rp_shutdown
Usage	Use with caution. No verification or authentication is performed before the server is shut down.
See also	rp_stop, rp_switch

rp_start

Description

Starts a group of previously stopped connections.

Note You can also use the OpenSwitch Manager to start connections to an Adaptive Server. See "Starting connections to Adaptive Server" on page 61 for more information.

Syntax

rp_start *pool_name*, *server*, *spid*

Parameters

pool_name

Name of the pool in which to restart connections. If you do not supply a parameter, or use NULL, all pools are started.

server

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.

spid

The OpenSwitch process ID of the connection to be started. If you do not supply an argument, or use NULL, all connections are started.

Examples

Example 1 Starts previously stopped *spid* number 8.

```
rp_start NULL, NULL, 8
```

Example 2 Starts all connections that are currently using remote server SYB_SERV1.

```
rp_start NULL, "SYB_SERV1", NULL
```

Example 3 Starts all connections established through POOL_A.

```
rp_start "POOL_A", NULL, NULL
```

Example 4 Starts all connections established to SYB_SERV1 through POOL_A.

```
rp_start "POOL_A", "SYB_SERV1", NULL
```

Usage

Attempting to start a connection that was not previously stopped has no effect.

Messages

- The *pool_name* does not exist. Use rp_pool_help to list valid pool names:

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

- The *server* does not exist:

  ```
  rp_start: Invalid server name 'server'.
  ```

Use rp_server_status to list valid servers.

See also rp_pool_help, rp_server_status, rp_stop, rp_who

rp_stop

Description

Stops a group of connections from issuing new queries.

Note You can also use the OpenSwitch Manager to stop connections to an Adaptive Server. See “Managing connections to Adaptive Server” on page 60 for more information.

Syntax

rp_stop [*pool_name*], [*server*], [*spid*], [*ign_tran*], [*ign_fail*]

Parameters

pool_name

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.

server

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.

spid

The OpenSwitch process ID of the connection to be stopped. If you do not supply an argument, or use NULL, all connections are stopped.

ign_tran

Indicates whether or not rp_stop ignores transaction state when pausing a connection.

Table 7-12: 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.

ign_fail

Indicates whether or not stopped connections ignore remote server failures while they are stopped.

Table 7-13: 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 *spid* 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.

Note You can also use the OpenSwitch Manager to switch connections between Adaptive Servers. See "Switching connections from Adaptive Server" on page 61 for more information.

Syntax

rp_switch [*pool_name*], [*src_server*], [*spid*], [*dst_server*], [*grace_period*], [*force*]

Parameters

pool_name

All connections established through the *pool_name* you specify are switched to the server specified by *dst_server*. If you do not supply this parameter, or specify NULL, all pools are assumed.

src_server

All the connections that are currently established to this remote server (*src_server*) switch to the *dst_server*. If you do not supply this parameter, or specify NULL, all servers are assumed.

spid

The OpenSwitch *spid* to be switched to the remote server *dst_server*. If you do not supply this parameter, or specify NULL, all *spids* are assumed.

dst_server

The remote server to which all connections identified by *pool_name*, *srv_server*, and *spid* 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.

Use care when specifying this parameter. No verification is performed for *dst_server*. Passing an invalid value causes all incoming client connections to be lost.

grace_period

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.

force

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.

Examples

Example 1 Causes OpenSwitch *spid* number 8 to be switched to the remote server SYB_SERV1. If the *spid* 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.

```
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.

Note You can also use the OpenSwitch Manager to set the trace flags options of your OpenSwitch servers. See "Editing OpenSwitch server properties" on page 51 for more information.

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.

Note You can also use the OpenSwitch Manager to view information about the OpenSwitch server. See "Viewing OpenSwitch server properties" on page 50 for more information.

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 information about user connections.

Note You can also use the OpenSwitch Manager to view information about user connections. See “Monitoring processes” on page 63 for more information.

Syntax

rp_who [*spid*]

Parameters

spid

Displays the value of the OpenSwitch *spid*.

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 7-14 describes the contents of each column returned by rp_who.

Table 7-14: 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 7-15 on page 238 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 rp_switch, but some of them may arise internally due to things like an Adaptive Server failing.Table 7-16 on page 238 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 current 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 7-15 describes the state of each connection returned by rp_who.

Table 7-15: Values for state

Value	Description
ATTNWAIT	Internal flag.
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 rp_switch on page 230).
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 dbcancel() or ct_cancel()). 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 rp_switch on page 230).

Table 7-16 describes the value for each action returned by rp_who.

Table 7-16: 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 rp_switch with a value of 1, or when the grace period has expired.

Value	Description
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.
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 rp_stop.
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

CHAPTER 8

Notification Procedures

This chapter describes notification registered procedures.

Introduction

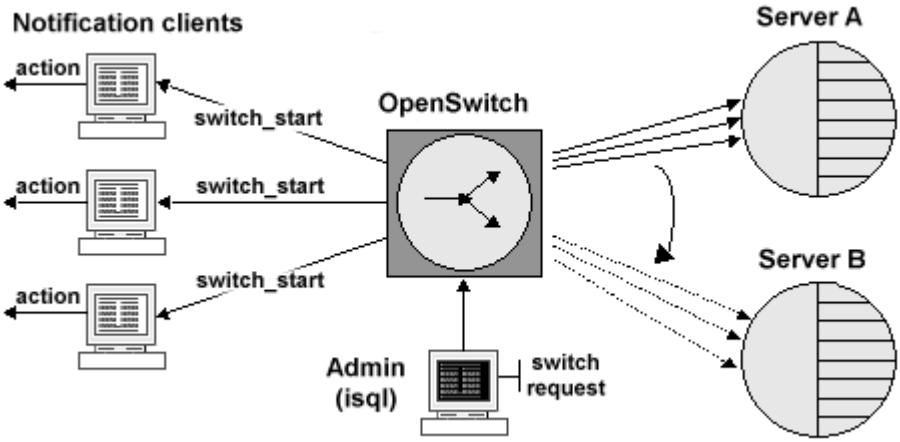
A notification is a special registered procedure that has no associated action or code, but can be used to notify Open Client applications when certain events occur within OpenSwitch.

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 8-1 on page 242.

Figure 8-1: 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 245 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 8-1: rsn_code valid values

Value	Description
1	The connection is logging in. *cur_server* is NULL, and *nxt_server* 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. *cur_server* contains the name of the server that failed, and *nxt_server* 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. *cur_server* contains the name of the failed server, and *nxt_server* 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. *cur_server* contains the name of the failed server, and *nxt_server* 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
spid	`11`
username	`sa`
appname	`ctisql`

Parameter	Example return value
hostname	`rhino`
database	`NULL`
pool	`pool1`
rsn_code	`1`
rsn_text	`Login attempt`
cur_server	`NULL`
nxt_server	`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. See the *OpenSwitch Coordination Module Reference Manual* for more information.

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	np_switch_end is broadcast when rp_switch completes. This does not indicate the success or failure of the procedure, only that is has completed.
See also	rp_switch, np_switch_start

Index

Symbols

A

B

C

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W