SYBASE®

Coordination Module Reference Manual

OpenSwitch

15.0

DOCUMENT ID: DC20190-01-1500-01

LAST REVISED: December 2005

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

Contents

About This Book

Audience

This book is for developers creating coordination modules for OpenSwitch version 15.0 and assumes that the reader has:

- A general knowledge of the operating system
- Familiarity with all platform-specific commands used to manipulate the software and hardware, such as those for changing directories and mounting the CD
- General knowledge of Sybase® servers
- General knowledge of failover systems
- In-depth knowledge of and experience with programming in the C language
- Basic knowledge of Sybase Open Client™ programming

How to use this book

This document includes these chapters:

- Chapter 1, "Introduction," describes coordination modules—user-built applications that connect to the OpenSwitch server and control client logins and failover patterns within OpenSwitch.
- Chapter 2, "Using Coordination Modules," describes the basic steps for building OpenSwitch coordination modules (CM) and provides example programs.
- Chapter 3, "Coordination Module Routines and Registered Procedures," provides a reference for each coordination module routine and registered procedure.
- Chapter 4, "Using the Replication Coordination Module," describes the sample replication coordination module (RCM) provided with OpenSwitch. You can use this Sybase-created RCM with Replication Server® in an OpenSwitch implementation.

Related documents

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

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

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

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

Other sources of information

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

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

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

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

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

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

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **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 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 formatting conventions used in this document are:

Formatting example	To indicate
command names and method names	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text
myCounter variable *Server*.log *myfile.txt*	Italic font indicates: • Program variables • Parts of input text that must be substituted • File names
sybase/bin	Directory names appearing in text display in lowercase unless the system is case sensitive. A forward slash ("/") indicates generic directory information. A backslash ("\") applies to Windows users only.
File | Save	Menu names and menu items display in plain text. The vertical bar indicates how to navigate menu selections, such as from the File menu to the Save option.
`create table` `table created`	Monospace font indicates: • Information that you enter on a command line or as program text • Example output fragments

Accessibility features

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

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

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

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

If you need help

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

CHAPTER 1

Introduction

This chapter introduces coordination modules (CMs), which are user-built applications that connect to an OpenSwitch server and control client logins and failover patterns within OpenSwitch.

Using coordination modules

The default behavior of OpenSwitch is to migrate failed client connections as they fail. For example, if a connection fails, OpenSwitch immediately migrates it to the next available Adaptive Server according to the mode of the pool in which the connection resides.

However, you may want to coordinate the switching process for certain OpenSwitch operations or business requirements. For example, when an Adaptive Server fails, you may want the client to reconnect to the failed server. Or, if a single connection fails unexpectedly, you may want to switch all connections to the next available server.

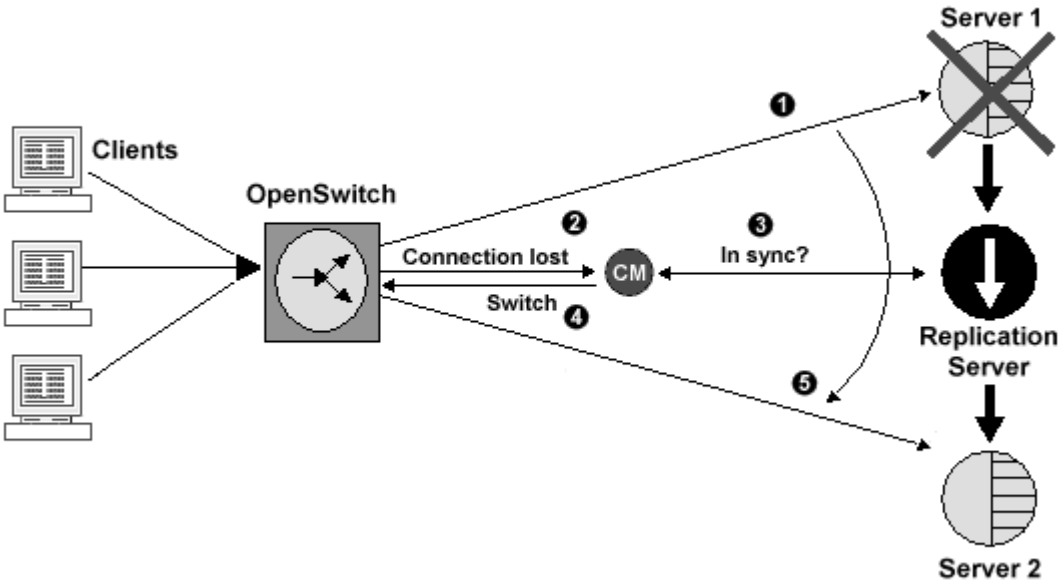
More importantly, you may need to coordinate the switching process with an external high-availability (HA) solution such as Sybase Replication Server®. In this case, failover should not occur until the HA service has completed the necessary steps to bring the backup server online, such as waiting until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

For these situations, OpenSwitch provides a simple application programming interface (API) that allows you to develop an external coordination module (CM). When connected to an OpenSwitch server, a coordination module receives event notifications based on connection state changes.

Note OpenSwitch provides a sample replication coordination module (RCM), which is a coordination module created using CM APIs. You can use the sample to coordinate failover of a high-availability, warm-standby system that uses Replication Server. See Chapter 4, “Using the Replication Coordination Module.”

For example, if a user attempts to log in, or a connection is lost to a server, the coordination module notifies OpenSwitch of the actions it should take, as illustrated in Figure 1-1.

Figure 1-1: 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 a connection is lost, the coordination module receives a message indicating which connection was lost, and the server with which that connection was communicating. The lost connection is suspended in the OpenSwitch server until the coordination module responds with the action to be taken for 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 can, at this point, attempt to automatically recover Server 1 before attempting to switch users to Server 2.

4 The coordination module responds to OpenSwitch that all connections that were using Server 1 should now switch to the next available 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 logins, you can also use the CM to override built-in OpenSwitch pooling and routing mechanisms with application- or business-specific logic.

The coordination module can:

- Determine if a failed connection is due to a remote Adaptive Server being unavailable
- Determine if the backup Adaptive Server is available
- Coordinate itself with third-party high-availability tools
- Switch all connections in tandem
- Mark an Adaptive Server as unavailable in OpenSwitch
- Manage multiple instances of OpenSwitch

If the OpenSwitch server 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.

Coordination modes

OpenSwitch runs in one of four coordination modes, determined by the value that you assign to the OpenSwitch configuration parameter *COORD_MODE*. The coordination mode specifies how OpenSwitch should respond in the presence of a coordination module. For details on using the configuration file, see Chapter 4, "Using the Configuration File," in the *OpenSwitch Administration Guide*.

The values for *COORD_MODE* are:

Mode	Description
NONE	OpenSwitch runs autonomously and makes all switching and routing decisions without a CM. Coordination modules can still connect to OpenSwitch, but do not receive notifications.
AVAIL	In the absence of a CM, OpenSwitch runs autonomously. If a CM is available and connected to the OpenSwitch server, the CM is used.
ALWAYS	A CM is required. If a CM is unavailable when a thread issues a request, the thread sleeps until a CM becomes available.
ENFORCED	A CM is required. If a CM is unavailable when a client makes a request, the request is refused and an informational message is sent back to the client.

Notification requests

When a client thread requests a response from a CM, the thread sleeps, or appears to have stopped responding to the client, until the thread receives a response from a coordination module.

A client connection is activated only by a CM response, or the client disconnecting before a CM response is issued. Coordination requests issued by threads are broadcast to all connected CMs via Open Client™ notification procedures.

Note A notification is a special registered procedure that has no associated action or code, but that can be used to notify Open Client applications when certain events occur within OpenSwitch. See Chapter 7, "Notification Procedures," in the *OpenSwitch Administration Guide*.

Notification procedures provide asynchronous communication with one or more client applications, which allows multiple CMs to be attached to an OpenSwitch server at any given time.

Note Each CM receives a copy of every notification broadcast. However, you must ensure that no more than one CM attempts to respond to any given message.

OpenSwitch uses an internal notification procedure, np_req_srv, to communicate with a CM and notify it of connections that are waiting for a response. This procedure is used by OpenSwitch internally to indicate that the connection is blocked and is awaiting a response from the CM, which can come in the form of a call to rp_set_srv, rp_switch, or rp_kill. Only these registered procedures (rp_set_srv, rp_switch, or rp_kill) or a disconnection from the client can "wake up" a connection waiting for a response.

Note np_req_srv is issued only if at least one CM is attached and the coordination mode is AVAIL, ALWAYS, or ENFORCED. For more information, see "Coordination modes" on page 4.

Coordination module responses

Coordination modules have no special response mechanism. The CM responds by issuing registered procedure calls, just as an OpenSwitch administrator would issue manually. Only a few registered procedures cause a thread to awaken after blocking a coordination request:

Response procedure	Description
rp_set_srv	A mirror of np_req_srv that responds to a specific OpenSwitch *spid* with the name of an Adaptive Server that OpenSwitch should use.
rp_switch	Similar to rp_set_srv, except you can use rp_switch to route multiple connections to another Adaptive Server.
rp_kill	Forcibly disconnects a client connection from OpenSwitch.

What if the coordination module is unavailable?

If a coordination module is unavailable and the coordination mode is ALWAYS, all client connections are refused until a coordination module becomes available. When the coordination module connects to the OpenSwitch server, all pending notifications are broadcast to the coordination module.

If a coordination module is unavailable, and the coordination mode is set to ENFORCED, the connection is refused and a message is sent back to the client.

If *COORD_MODE* is set to AVAIL, client connections are made if a coordination module is available. If a coordination module is not available, OpenSwitch requests the name of an available Adaptive Server from the defined pools in the configuration file.

CHAPTER 2

Using Coordination Modules

This chapter explains how to build an OpenSwitch coordination module. Example programs are provided.

Introduction

A coordination module connects to an OpenSwitch server to control client logins and failover patterns within OpenSwitch. 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. The OpenSwitch installation provides the APIs needed to create a CM, including:

- *libcm.so* (on Solaris, RS6000, DEC, and Linux), *libcm.sl* (on HP-UX), or *libcm.lib* (on Windows) – located in *$OPENSWITCH/lib* on UNIX and *%OPENSWITCH%\lib* on Windows, this is the library that contains all the CM API definitions.
- *cm.h* – located in *$OPENSWITCH/include* on UNIX and *%OPENSWITCH%\include* on Windows, this is the header file that contains the prototype declarations for all the CM APIs.
- Open Client libraries – located in *$SYBASE/$SYBASE_OCS/lib* on UNIX or *%SYBASE%\%SYBASE_OCS%\dll* on Windows.

- Open Client header files – located in *$SYBASE/$SYBASE_OCS/include* on UNIX and *%SYBASE%\%SYBASE_OCS%\include* on Windows.
- *cm1.c* – located in *$OPENSWITCH/sample* on UNIX and *%OPENSWITCH%\sample* on Windows, this is a sample CM program, complete with a *README* and *Makefile*.

Compiling the coordination module

Use the *Makefile* located in *$OPENSWITCH/sample* on UNIX and *%OPENSWITCH%\sample* on Windows to compile your CM program.

1 With a text editor, open *Makefile* and replace "cm1" with the name of your CM program.

2 Enter the following, where <*CMsource*> is the directory containing your CM source code.

On UNIX:

```
source $SYBASE/SYBASE.csh
cp $OPENSWITCH/sample/Makefile <CMsource>
```

On Windows in a Command Prompt window:

```
%SYBASE%\SYBASE.bat
cp %OPENSWITCH%\sample\Makefile <CMsource>
```

Creating a minimal coordination module

These basic steps allow you to build a minimal CM library program that establishes a connection to OpenSwitch:

1 Allocate a context structure using cm_init.

2 Create a CM instance using cm_create.

3 Establish connection between the CM and Open Switch using cm_connect.

4 Start the CM using cm_run.

5 Destroy the CM instance using cm_destroy.

6 Deallocate the context structure using cm_exit.

For details about the routines used to build a CM, see Chapter 3, “Coordination Module Routines and Registered Procedures.”

The following example program shows the steps required to create a minimal CM.

```
#include <stdio.h>
#include <string.h>
#include <cspublic.h>
#include <cm.h>

int
main (
CS_INT argc,
CS_CHAR *argv[]
)  {

char *username = "switch_coord";
char *password = "switch_coord";
char *server = "SWITCH1";

   cm_ctx_t  *ctx;
   cm_t      *cm;

   /* Initialize and allocate a coordination module context structure
   **for this executable.
   */
   if (cm_init(&ctx) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_init: Failed.\n");
      exit (1);
   }

   /* Create a coordination module instance to manage an OpenSwitch server.
   */
   if (cm_create(ctx, &cm) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_create: Failed.\n");
   cm_exit (ctx);
   }

   /* Establish a connection between the coordination module and a single
   **OpenSwitch server.
   */
   if (cm_connect(cm, server, username, password)
!= CS_SUCCEED)
   {
```

```
      fprintf (stderr, "cm_connect: Unable to connect to server\n" );
      cm_destroy (cm);
      cm_exit (ctx);
      exit (1);
   }

   /* Start the coordination module.
   */
   if (cm_run(ctx) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_run: Failed.\n");
   }

   /* Destroy the coordination module instance.
   */
   cm_destroy (cm);

   /* Deallocate the coordination module instance and Exit.
   */
   cm_exit (ctx);
   exit (0);
}
```

Installing a callback handler

After compiling and running the CM, you must install callback handlers so the CM can detect connection requests coming in from OpenSwitch, and handle them accordingly.

In this example, the CM from the previous example is expanded to include a callback handler, which handles a dropped connection between the CM and OpenSwitch. You must call cm_set_prop to allow asynchronous callbacks; you must call cm_callback to install the callback handler.

This example shows the steps required to install the callback handler.

```
#include <stdio.h>
#include <string.h>
#include <cspublic.h>
#include <cm.h>

CS_STATIC CS_RETCODE CS_PUBLIC cm_lost_hdl();

int
main (
```

```
CS_INT argc,
CS_CHAR *argv[]
)  {

char *username = "switch_coord";
char *password = "switch_coord";
char *server = "SWITCH1";

   cm_ctx_t  *ctx;
   cm_t      *cm;

   /* Initialize and allocate a coordination module context structure

   **for this executable.
   */
   if (cm_init(&ctx) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_init: Failed.\n");
      exit (1);
   }

   /* Create a coordination module instance to manage an OpenSwitch server.
   */
   if (cm_create(ctx, &cm) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_create: Failed.\n");
      cm_exit (ctx);
   }

   /* Allow asynchronous callbacks
   */
   if (cm_set_prop (cm, CM_P_ASYNC,
(CS_VOID*)&async) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_callback: Unable to set async property\n");
      cm_destroy (cm);
      cm_exit (ctx);
   }

   /* Install the connection lost callback handler.
   */
   if (cm_callback (cm, CM_CB_LOST,
(CS_VOID*)cm_lost_hdl) != CS_SUCCEED)
   {
      fprintf(stderr, "cm_callback: Unable to install CM_CB_LOST handler\n");
      cm_destroy (cm);
      cm_exit (ctx);
      exit(1);
```

```
   }

   /* Establish a connection between the coordination module and a single
   ** OpenSwitch server.
   */
   if (cm_connect(cm, server, username, password)!= CS_SUCCEED)
   {
      fprintf (stderr, "cm_connect: Unable to connect to server\n" );
      cm_destroy (cm);
      cm_exit (ctx);
      exit (1);
   }

   /* Start the coordination module.
   */   if (cm_run(ctx) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_run: Failed.\n");
   }

   /* Destroy the coordination module instance.
   */
   cm_destroy (cm);

/* De-allocate the coordination module instance and Exit.
*/
cm_exit (ctx);

exit (0);

}

/*
* cm_lost_hdl():
*
   /* This is a coordination module handler function that is called every
   ** time the connection is lost to OpenSwitch managed by cm. It is responsible
   ** for installing a timer callback that will attempt to reconnect to
   ** the OpenSwitch every 10 seconds (see cm_time_connect()).
*/
CS_STATIC CS_RETCODE CS_PUBLIC cm_lost_hdl (
cm_t *cm
) {
   fprintf (stdout, "**** Connect Lost ******\n");

   if (cm_timer_add (cm, "LOST_TIMER", (CS_INT)10000,
      (cm_timer_cb*)cm_time_connect, (CS_VOID*)NULL, (CS_INT)0 ) != CS_SUCCEED)
   {
      fprintf (stderr, "cm_lost_hdl: Unable to add cm_time_connect\n");
      return CS_FAIL;
```

```
   }

   return CS_SUCCEED;
}

/*
* cm_time_connect():
*
   ** This function is installed as a timed callback from a CM_CB_LOST
   ** callback handler. After it is installed, this function is called
   ** periodically to attempt to re-establish the coordination
   ** module's connection to its OpenSwitch. After the connection
   ** is re-established, this timer removes itself from activity.
*/
CS_STATIC CS_RETCODE CS_PUBLIC cm_time_connect(
   cm_t    *cm,
   CS_CHAR *name,
   CS_VOID *data
) {
   fprintf (stdout, "cm_time_connect: Attempting re-connect...\n");

   if (cm_connect( cm, NULL, NULL, NULL ) == CS_SUCCEED)
   {
      if (cm_timer_rem( cm, name ) != CS_SUCCEED)
      {
         fprintf( stderr, "cm_time_connect: Unable to remove timer\n" );
         return CS_FAIL;
      }
   }
   return CS_SUCCEED;
}
```

After installing the callback handler, the CM in this example immediately detects when the OpenSwitch server goes down and starts a timer to ping the OpenSwitch server every ten seconds until the OpenSwitch server comes back online. If you want the CM to also respond to logins and failovers from the OpenSwitch server, you must create a complete CM such as the one in the next section.

Creating a complete coordination module

OpenSwitch provides a sample of a self-contained CM that coordinates the activities of an OpenSwitch server and demonstrates most of the calls in *libcm*. See the sample code file, which is installed along with OpenSwitch, in *$OPENSWITCH/sample/cml.c* on UNIX and in *%OPENSWITCH%\sample\cml.c* on Windows.

The CM in the sample file responds to login attempts and login retries.

Event	Response
Login attempt	Login is allowed through the server that was automatically chosen by OpenSwitch.
Login retry	The Adaptive Server® that the failing *spid* is attempting to connect to is pinged. If the server is available, the *spid* is killed. If the server cannot be pinged, the connection tries the next server.

Enabling Sybase Failover

To support Sybase Failover, add code to the existing CMs. See the *cml.c* sample in *$OPENSWITCH/sample/* on UNIX and *%OPENSWITCH%\sample* on Windows, under the cm_srvreq_hdl function under the case for *COORD_R_HAFAILOVER*. Add this code to the CM:

```
case COORD_R_HAFAILOVER:
      if (cm_set_srv(cm, req->spidm req->cur_server) !=CS_SUCCEED
      {
              fprintf(stderr, "cm1: Unable for spid '%d' to stay put on the
                     current server\n", (int) req->spid);
       }
        break;
```

This code segment returns the server name. In a multicluster environment, the CM is notified of a failover event only when the entire primary Adaptive Server Enterprise cluster fails. The CM is not notified when the primary node of the Adaptive Server cluster fails, as connections are automatically redirected to the secondary node without consulting the CM.

Using concurrent coordination modules

OpenSwitch allows you to run multiple CMs concurrently against one OpenSwitch server to create a redundant environment where a CM is not a single point of failure.

Note The RCM does not support concurrent coordination modules. When the RCM establishes a connection to OpenSwitch, OpenSwitch sets the *COORD_TIMEOUT* to zero (0), which turns off the coordinated CM functionality. *COORD_TIMEOUT* must be set to zero (0) for the RCM to start.

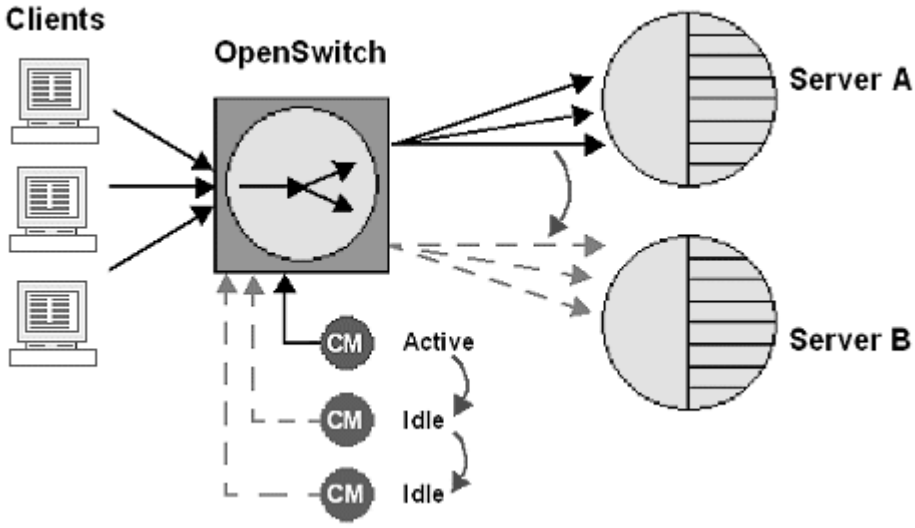
When multiple CMs connect to one OpenSwitch, the following activities occur, which are transparent to the end user:

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 (CM ID) and sends that CM ID 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 for the *COORD_TIMEOUT* configuration parameter, OpenSwitch retrieves the next CM ID from the internal list of inactive CMs. All future notifications include the new CM ID as part of the notification.

Figure 2-1: Concurrent coordination modules



Note See “Creating a minimal coordination module” on page 8 for basic instructions on creating CMs.

Configuration

Use the *COORD_TIMEOUT* parameter in the [CONFIG] section of the OpenSwitch server configuration file to specify the number of seconds OpenSwitch waits for a response before determining that a CM is not responding. The default *COORD_TIMEOUT* value is zero (0) seconds.

Note If you set *COORD_TIMEOUT* value to zero in the OpenSwitch configuration file, concurrent CMs are not used.

In a legacy, pre-15.0 CM application, you must set *COORD_TIMEOUT* to zero (0) in the [CONFIG] section of the OpenSwitch configuration file, or the CM receives an error message and does not start.

If a CM does not respond within the *COORD_TIMEOUT* period, the OpenSwitch server acquires the next CM ID from the internal list of inactive CMs. The previous CM ID becomes obsolete. If the CM becomes active again, it requests a new CM ID from the OpenSwitch. This occurs in the CM API and is transparent to applications.

If a connection is lost because OpenSwitch fails for some reason, the callback handler for *COORD_CB_LOST* is called for any CM that has lost the connection. When OpenSwitch restarts, if the CMs that lost their connection detect that the OpenSwitch has restarted, those CMs reconnect and are issued a new CM ID on a first-come basis.

Notifications

When multiple CMs are connected to OpenSwitch and *COORD_TIMEOUT* is set to zero (that is, OpenSwitch is not configured to use concurrent CMs), all CMs are registered to receive notifications sent by the OpenSwitch server. The OpenSwitch server handles the first instance of a response to a notification; all other instances of the same notification yield an error in the OpenSwitch server's log file.

Enabling mutually-aware support

OpenSwitch 15.0 and later includes the coordination module API cm_get_value, which you can use to retrieve the mutually-aware configuration value of an OpenSwitch server.

See Chapter 5, "Using Mutually-aware OpenSwitch Servers," in the *OpenSwitch Administration Guide* for details about configuring mutually-aware support.

cm_get_value

Description

Because mutually-aware OpenSwitch servers do not currently support removing or adding Adaptive Servers to pools, before adding or removing a server, this API is used to retrieve the mutually-aware configuration value of of an OpenSwitch server.

If you select Use Mutual Aware Support? in the configuration GUI (`MUTUAL_AWARE=1` in the OpenSwitch configuration file), servers can neither be added or removed from a pool.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_set(*cm*, *parm_name*, *parm_value*)

cm_t **cm*;

CS_CHAR **parm_name*;

CS_CHAR **parm_value*;

Parameters

- *cm* – pointer to a CM control structure.
- *parm_name* – name of a configuration variable as listed in the configuration file.
- *parm_value* – *cm_get_value* returns the value of the configuration parameter specified for *parm_value*.

Example

```
If(cm_get_value( cm, "DEBUG",  parm_val ) !=CS_SUCCED)
{
     cm_error("Unable to retrieve the value of the 'DEBUG' configuration
               parameter\n");
     return  CS_FAIL;
}
```

Enabling encryption

Two APIs are available for CM applications to support encryption:

- cm_connect_enc
- cm_ping_enc

The *cm1.c* sample, located in *$OPENSWITCH/sample* on UNIX and in *%OPENSWITCH%\sample* on Windows, has been modified to use these APIs. Use the -E flag to specify that the user names and passwords are encrypted. You can use a shell script to invoke "cm1" using the encrypted user name/password combinations. For example:

```
#!/usr/bin/sh
./cm1 \
-U 0x010c7ec... \
-P 0x010c7ec... \
-u 0x102c06... \
```

```
-p 0x102dcd... \
-S OSWITCH1 -E
```

You need not follow this convention in your CM applications. Sybase recommends that if you choose to enforce encryption of user names and passwords, that you set the encryption argument to `CS_TRUE` in both cm_connect_enc and cm_ping_enc. See Chapter 3, “Coordination Module Routines and Registered Procedures.”

The *cm1.c* sample allows you to use either encrypted or unencrypted values depending on the arguments you pass. See the *OpenSwitch Administration Guide* for more information about encryption support.

CHAPTER 3

Coordination Module Routines and Registered Procedures

This chapter describes the routines and registered procedures that you can call within a coordination module (CM).

Coordination module routines

Table 3-1 lists the routines used by coordination modules.

Table 3-1: CM routines

Routine	Description
cm_callback	Installs or removes a CM event callback handler.
cm_close	Closes an established connection between a CM and OpenSwitch.
cm_connect	Establishes a connection between a CM and OpenSwitch.
cm_connect_enc	Allows the use of encrypted user names and passwords when making a connection.
cm_create	Creates a CM instance.
cm_destroy	Destroys a CM instance.
cm_error	Outputs an error message.
cm_exit	Exits and unallocates CM context.
cm_getopt	Parses command line arguments.
cm_get_prop	Retrieves a property of a CM.
cm_get_value	Retrieves the mutually-aware configuration value of an OpenSwitch server. See the *OpenSwitch Administration Guide* for details about using mutually-aware OpenSwitch servers.
cm_ignore	Ignores OpenSwitch messages matching a given template.
cm_ignore_clear	Sets all fields of a message structure to empty values.
cm_init	Initializes a CM instance.
cm_optreset	Resets the state of option parsing for cm_getopt.
cm_ping	Verifies the health of a remote server.
cm_ping_enc	Allows the use of encrypted user names and passwords when pinging.

Routine	Description
cm_repeat_ping	Verifies the health of a remote server, repeating if a failure occurs.
cm_repeat_short_ping	Sets a time limit on the duration each ping waits when a failure occurs.
cm_run	Starts the CM.
cm_set_print	Installs an error display function.
cm_set_prop	Sets a property of a CM.
cm_short_ping	Sets a time limit on the number of seconds allowed for the CM to establish its connection.
cm_start	Resumes activity of connections.
cm_stop	Suspends activity of connections.
cm_timer_add	Adds a timed callback.
cm_timer_rem	Removes a timed callback.
cm_unignore	Removes OpenSwitch ignore requests matching template.
cm_version	Returns the pointer to the location of the version string.

cm_callback

Description — Installs or removes a CM event callback handler.

Syntax

CS_RETCODE cm_callback(*cm*, *cb_type*, *cb_func*)
cm_t **cm*;
CS_INT *cb_type*;
CS_VOID **cb_func*;

Parameters

cm

A pointer to a CM control structure.

cb_type

The CM event callback handler being installed. Valid values for *cb_type* are:

Callback type	Description
CM_CB_CTLIB	Called each time an Open Client API error message is generated. If not defined, these messages display to *stderr* when they are received. Use the cm_set_print function to overwrite this behavior. Equivalent to an Open Client CS_CLIENTMSG_CB command.
CM_CB_LOST	Called by a CM to a remote OpenSwitch server from which the connection is lost. If not defined, these messages are ignored.
CM_CB_MSG	Called each time a message is received from OpenSwitch. If not defined, these messages are displayed to *stderr* when they are received. Use the cm_set_print function to overwrite this behavior. Equivalent to an Open Client CS_SERVERMSG_CB command.
CM_CB_SERVER	Called by a client connection, requesting the name of a remote server to either log in to or switch to. If not defined, these messages are ignored.

cb_func

A pointer to a function to be called when a message of *cb_type* is received. Valid values for *cb_func* are:

Callback type	Description	Form
CM_CB_CTLIB	Open Client message callback handler	`cb_func(CS_CONTEXT *context, CS_CONNECTION *connection, CS_CLIENTMSG *clientmsg, cm_t *cm)`
CM_CB_LOST	Connection lost message	`cb_func(cm_t *cm)`
CM_CB_MSG	Server message callback handler	`cb_func(CS_CONTEXT *context, CS_CONNECTION *connection, CS_SERVERMSG *servermsg, cm_t *cm)`
CM_CB_SERVER	Server request message callback	`cb_func(cm_t *cm, cm_req_srv_t *req`

Return value — cm_callback returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
static CS_RETCODE cm_lost_handler( cm )
   cm_t *cm;
{
   fprintf( stderr, "Connection lost!" );
   return CS_SUCCEED;
}

main()
{
   CS_RETCODE retcode;
   ...

   retcode = cm_callback( cm, CM_LOST_CB,
      (CS_VOID*)cm_lost_handler );

   if (retcode != CS_SUCCEED)
   {
      fprintf( stderr, "cm_callback failed!" );
      exit(1);
   }
   ...
}
```

Usage

- When you create a CM with cm_create, the CM has no callback handlers installed. The default callback actions are performed as described in the Parameters section.
- Unlike Open Client, you cannot establish callbacks at the CM context level, so callbacks are not inherited between modules or the context, and must be explicitly set for each module. For more information, see “cm_init” on page 45.
- To uninstall an existing callback, program an application to call cm_callback with *cb_func* as NULL. You can install a new callback any time the application is running. New callbacks automatically replace existing callbacks.
- Program an application to use the *CM_P_USERDATA* property to transfer information to a callback routine and the program code that triggered it. The *CM_P_USERDATA* property allows an application to save user data in internal space and retrieve it later.

- If the CM process exits for any reason, such as the OpenSwitch server failing, program the callback to return CS_FAIL to its caller. This return status is necessary for the CM to perform the necessary cleanups before the process exits.

See also cm_init, cm_create

cm_close

Description

Closes an established connection between a CM and OpenSwitch.

Syntax

```
CS_RETCODE cm_close(cm)
    cm_t *cm;
```

Parameters

cm

A pointer to a CM control structure.

Return value

cm_close returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Usage

- Closes an existing connection between a CM and a remote OpenSwitch using the cm_connect function.
- It is not an error to close a connection that was never opened; that is to say, if cm_connect was never called or has already been closed due to another event, for example, OpenSwitch unexpectedly failing.
- Closing the connection associated with a CM does not destroy the CM instance. Use cm_destroy to destroy the CM instance.

See also

cm_connect, cm_destroy

cm_connect

Description

Establishes a connection between a CM and OpenSwitch.

Syntax

CS_RETCODE cm_connect(*cm*, *server*, *username*, *password*)
cm_t **cm*;
CS_CHAR **server*;
CS_CHAR **username*;
CS_CHAR **password*;

Parameters

cm

A pointer to a CM control structure.

server

A pointer to the name of the OpenSwitch server to which to connect. *server* is the name of the server's entry in the *sql.ini* file on Windows and in the *interfaces* file on UNIX, or in the directory services source. A NULL *server* value may be supplied only if cm_connect has successfully attached to a remote server in the past. For more information, see the Usage section for this routine.

username

The name to be used to connect to OpenSwitch. This should match the *COORD_USER* configuration value in the OpenSwitch configuration file. For more information, see the *OpenSwitch Administration Guide*. A NULL *username* value may be supplied only if cm_connect has successfully attached to a remote server in the past.

password

The OpenSwitch user password to be used to connect to OpenSwitch. This value should match the COORD_PASSWORD configuration value in the OpenSwitch configuration file. For more information, see the *OpenSwitch Administration Guide*. A NULL *password* value may be supplied only if cm_connect has successfully attached to a remote server in the past.

Return value

cm_connect returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
cm_t *cm;
/*
* Create a new coordination module.
*/
cm = cm_create( ... )
```

```
...
if (cm_connect( cm, "SYB_SWITCH1", "coord_user",
   "coord_password" ) != CS_SUCCEED)
{
   fprintf( stderr, "cm_connect failed!\n" );
   return CS_FAIL;
}
```

Usage

- The cm_connect function is used to connect an instance of a CM to a remote OpenSwitch server. The *username* and *password* parameters are used by the CM to identify itself to OpenSwitch. Supplying a *username* and *password* that do not match the *COORD_USER* and *COORD_PASSWORD* configuration parameters in OpenSwitch causes cm_connect to return CS_FAIL.
- Internally, cm_connect establishes an Open Client connection to the OpenSwitch server, and waits for an acknowledgment by OpenSwitch that the appropriate *username* and *password* have been supplied. After connecting, cm_connect registers itself to be aware of several notification procedures, in particular, np_req_srv. For details about np_req_srv, see the *OpenSwitch Administration Guide*, Chapter 7, “Notifications Procedures.”
- Issuing a call to cm_connect while a connection is already established closes the existing connection (internally, using cm_close) before the new connection is attempted.
- If *cm* has been successfully connected to a server in the past using cm_connect, then passing a NULL value for any one of *server*, *username*, and *password* causes the value passed during the previous call to cm_connect to be used.

See also

cm_create, cm_close, cm_connect_enc

cm_connect_enc

Description

Similar to cm_connect, except it allows for the use of encrypted user names and passwords.

Syntax

```
CS_RETCODE CS_PUBLIC cm_connect(cm, server, username,
        password, encrypted)
    cm_t *cm
    CS_CHAR *server
    CS_CHAR *username
    CS_CHAR *password
    CS_BOOL encrypted
```

Parameters

cm

Pointer to a CM control structure.

server

A pointer to the name of the OpenSwitch server to which to connect. *server* is the name of the server's entry in the *$SYBASE/interfaces* file on UNIX, the *%SYBASE%\sql.ini* file on Windows, or directory services source. A NULL *server* value may be supplied only if cm_connect_enc has successfully attached to a remote server in the past.

username

The name to be used to connect to OpenSwitch. This should match the *COORD_USER* configuration value in the OpenSwitch configuration file. For more information, see the *OpenSwitch Administration Guide*. A NULL *username* value may be supplied only if cm_connect_enc has successfully attached to a remote server in the past.

password

The OpenSwitch user password to be used to connect to OpenSwitch. This value should match the *COORD_PASSWORD* configuration value in the OpenSwitch configuration file. For more information, see the *OpenSwitch Administration Guide*. A NULL *password* value may be supplied only if cm_connect_enc has successfully attached to a remote server in the past.

encrypted

A Boolean value that defines whether the user name and password are encrypted or not. If encrypted is set to CS_TRUE, all user names and passwords passed to the API must be encrypted. If set to CS_FALSE, none of the user names and passwords should be encrypted.

Return value

cm_connect_enc returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.

Return value	Meaning
CS_FAIL	The routine failed.

Examples

```
cm_t *cm;
/*
* Create a new coordination module.

*/
cm = cm_create( ... )

...
if (cm_connect_enc( cm, "SYB_SWITCH1",
"0x010a60c07b7f86c1d30fac6162ce70400daecdd6749335832fd
5c9c613e95ef6","0x010ed474cfcb327562ac19d5c6cad2f04733
e321d8983d474744ec3b80888bc0", 1) != CS_SUCCEED)
{
   fprintf( stderr, "cm_connect_enc failed!\n" );
   return CS_FAIL;
}
```

Usage

- Similar to cm_connect with the additional ability to pass in encrypted *username* (*COORD_USER*) and *password* (*COORD_PASSWORD*).
- If encryption is set to true, both *username* and *password* must be in encrypted form, and must also be encrypted in the OpenSwitch server.

See also

cm_connect

cm_create

Description

Creates a CM instance.

Syntax

CS_RETCODE cm_create(*ctx*, *cm*)
cm_ctx_t **ctx*;
cm_t **cm*;

Parameters

ctx

Pointer to a CM context structure. This context must be allocated and initialized by cm_init prior to calling cm_create.

cm

The address of a pointer variable. cm_create sets *cm* to the address of a newly allocated cm_t structure.

Return value

cm_create returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully. *cm* contains a pointer to a new cm_t structure.
CS_FAIL	The routine failed. The contents of *cm* are undefined.

Examples

```
cm_t   *cm;

/*
* Create a coordination module context.
*/
...

if (cm_create( ctx, &cm ) != CS_SUCCEED)
{
   fprintf( stderr, "cm_create() failed!\n" );
   return CS_FAIL;
}
```

Usage

- cm_create allocates a new CM to manage a single OpenSwitch server. This CM does nothing until callback handlers are installed using cm_callback and the CM is connected to an OpenSwitch using cm_connect.
- The *ctx* acts as a container for all CMs created with cm_create. This structure may be used to represent a self-contained group of CMs.

See also

cm_connect, cm_run, cm_callback, cm_init

cm_destroy

Description

Destroys a CM instance.

Syntax

CS_RETCODE cm_destroy(*cm*)
cm_t **cm*;

Parameters

cm

A pointer to a CM control structure to be destroyed.

Return value

cm_destroy returns these values:

Return value	Meaning
`CS_SUCCEED`	The routine completed successfully.
`CS_FAIL`	The routine failed.

Examples

```
if (cm_destroy( cm ) != CS_SUCCEED)
{
   fprintf( stderr, "cm_destroy() failed!\n" );
   return CS_FAIL;
}
```

Usage

- cm_destroy frees all resources associated with an instance of a CM cm_t structure. All memory used by the structure is reclaimed, and any active connection to an OpenSwitch server is closed.
- After a cm_t structure has been destroyed, it cannot be reused. A new structure must be allocated with cm_create.

See also

cm_create

cm_error

Description

Prints an error message to *stderr*.

Syntax

CS_VOID cm_error(*fmt*, ...)
CS_CHAR **fmt*;

Parameters

fmt

An output format string. This string may contain all of the output format specifications accepted by `fprintf(3c)`.

Return value

None.

Examples

```
cm_error( "Could not open file '%s': %s\n",
   (char*)file_name,
   strerror( errno ) );
```

Usage

- The cm_error function is identical to the standard C printf function. It formats the output according to the *fmt* string and prints it, by default, to *stderr*.
- To print an error message to a file, use the cm_set_print function instead.

See also

cm_set_print

cm_exit

Description

Exits and unallocates CM context.

Syntax

CS_RETCODE cm_exit(*ctx*)
cm_ctx_t **ctx*;

Parameters

ctx

A pointer to the coordination context structure to be destroyed.

Return value

cm_exit returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_exit( ctx ) != CS_SUCCEED)
{
   cm_error( "Unable to destroy context\n" );
   return CS_FAIL;
}
```

Usage

- A coordination context is used to encapsulate multiple OpenSwitch connections.
- *ctx* must point to a valid coordination context structure allocated using cm_init.
- Attempting to call cm_exit while any CMs exist within the context returns an error. cm_destroy must be used to destroy existing CMs prior to calling cm_exit.

See also

cm_init, cm_destroy

cm_getopt

Description

Parses command line arguments.

Syntax

CS_INT cm_getopt(*argc*, *argv*, *optstring*)
CS_INT *argc*;
CS_CHAR **argv*[ ];
CS_CHAR **optstring*;

Parameters

argc

The number of arguments held in the command line vector *argv*.

argv

Command line argument vector containing arguments.

optstring

Contains the option letters recognized by the command using cm_getopt. If a letter is followed by a colon, the option is expected to have an argument. If the letter is followed by a semicolon, an option is allowed but not required. If there is no character after the letter, an option is not allowed.

Return value

cm_getopt returns these values:

Return value	Meaning
'?'	An invalid option was supplied.
EOF	The last option was processed.
char	The command line option parsed.

Examples

```
main( argc, argv ) int   argc;
   char *argv[];
{
   extern CS_INT cm_optind;
   extern CS_INT cm_optarg;

   CS_INT   c;
   CS_INT   aflg = 0;
   CS_INT   bflg = 0;
   CS_INT   errflg = 0;
   CS_CHAR *ofile = NULL;

   while ((c = cm_getopt(argc, argv,
      "abo:")) != EOF)
   {
      switch(c)
      {
         case 'a':
            if (bflg)
               errflg++;
```

```
                else
                   aflg++;
                break;
             case 'b':
                if (aflg)
                   errflg++;
                else
                   bflg++;
                break;
             case 'o':
                ofile = cm_optarg;
                printf("ofile = %s\n", ofile);
                break;
             case '?':
                errflg++;
          }

          if (errflg)
          {
             fprintf(stderr,
                "usage: cmd [-a|-b] [-o "
                "<filename>] files...\n" );
             exit (2);
          }

          for (; cm_optind < argc; cm_optind++)
          {
             printf("%s\n", argv[cm_optind]);
             return 0;
          }
    }
```

The code fragment shows how to process the arguments for a command that can take the mutually exclusive options `a` and `b`, and the option `o`, which requires an argument:

Usage

- cm_getopt returns the next option letter in *argv* that matches a letter in *optstring*.
- If an option accepts an argument (the option letter is followed by a colon or a semicolon in *optstring*), the contents of the argument are found in the global variable cm_optarg. If an argument is optional and is not supplied, cm_optarg is NULL.

- cm_getopt places in the cm_optind the *argv* index of the next argument to be processed. cm_optind is external and is initialized to 1 before the first call of cm_getopt. When all options have been processed, up to the first nonoptional argument, cm_getopt returns `EOF`.
- The cm_optreset function may be used to reset the state of cm_getopt.

See also cm_getopt, cm_optreset

cm_get_prop

Description

Retrieves a property of a CM.

Syntax

CS_RETCODE cm_get_prop(*cm*, *prop*, *value*)
cm_t **cm*;
CS_INT *prop*;
CS_VOID **value*;

Parameters

cm

A pointer to a CM control structure.

prop

The name of the property to be retrieved. Valid values for *prop* are:

Callback type	Description
CM_P_ASYNC	Checks to see if asynchronous notification is set in the connection between the CM and OpenSwitch.
CM_P_USERDATA	Retrieves a pointer from the CM control structure previously attached using cm_set_prop(CM_P_USERDATA). This property may be used to pass data between CM callbacks.
CM_P_NAME	Retrieves the name of the OpenSwitch server to which the CM is connected. If the module has never been connected, an empty string is returned.

value

A pointer to a memory location in which the CM property is retrieved. Valid values for *value* are:

Callback type	Value
CM_P_ASYNC	A pointer to a CS_BOOL value.
CM_P_USERDATA	A pointer to a void pointer (CS_VOID**).
CM_P_NAME	A pointer to an array of CS_CHAR of length 31 or greater.

Return value

cm_get_prop returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
CS_CHAR  cm_name[31];

if (cm_get_prop( cm, CM_P_NAME, cm_name )
   != CS_SUCCEED)
{
   cm_error("Unable to retrieve CM_P_NAME prop \n");
```

```
    return CS_FAIL;
}
else
{
    fprintf( stdout,
    "module name: %s\n", (char*)cm_name );
}
```

Usage

Although the contents of all CM data structures are transparent (versus opaque), fields within the data structures should never be accessed directly. Instead, the cm_get_prop or cm_set_prop functions should be used. This allows the internal definitions to be changed in future releases without affecting existing code.

See also

cm_set_prop

cm_get_value

Description

Because mutually-aware OpenSwitch servers do not currently support removing or adding Adaptive Servers to pools, before adding or removing a server, use this API to retrieve the mutually-aware configuration value of an OpenSwitch server.

If you select Use Mutual Aware Support? in the configuration GUI (MUTUAL_AWARE=1 in the OpenSwitch configuration file), servers can neither be added or removed from a pool.

Syntax

CS_RETCODE CS_PUBLIC cm_get_value(*cm*, *parm_name*, *parm_value*)

cm_t **cm*;

CS_CHAR **parm_name*;

CS_CHAR **parm_value*;

Parameters

cm

A pointer to a CM control structure.

parm_name

Name of a configuration variable as listed in the configuration file.

parm_value

Returns the value of the configuration parameter specified for *parm_value*.

Examples

```
If(cm_get_value( cm, "DEBUG",  parm_val ) !=CS_SUCCEED)
```

```
{
     cm_error("Unable to retrieve the value of the 'DEBUG' configuration
               parameter\n");
     return  CS_FAIL;
}
```

cm_ignore

Description

Ignores OpenSwitch messages matching a given template to prevent it from invoking the corresponding callback handler as installed by cm_callback.

Syntax

CS_RETCODE cm_ignore(*cm*, *msg_type*, *msg*)
cm_t **cm*;
CS_INT *msg_type*;
CS_VOID **msg*;

Parameters

cm

A pointer to a CM control structure.

msg_type

The type of message being passed through the *msg* argument. The only valid value for *msg_type* is:

Callback type	Description
CM_CB_SERVER	A server-name request message

msg

A pointer to a cm_req_srv_st structure, which is defined as:

```
typedef struct cm_req_srv_st {
    CS_INT     spid; /* OpenSwitch spid number */
    CS_CHAR    username[31]; /* Name of the user */
    CS_CHAR    appname[31]; /* Application they are running */
    CS_CHAR    hostname[31]; /* Host client is running on */
    CS_CHAR    database[31]; /* Host client is running on */
    CS_CHAR    pool[31]; /* Pool the user is in */
    CS_INT     rsn_code; /* Reason for request (see cm_rsn.h) */
    CS_CHAR    rsn_text[256]; /* Description of reason */
    CS_CHAR    cur_server[31]; /* Current server they are using */
    CS_CHAR    nxt_server[31]; /* Server they want to go too */
} cm_req_srv_t;
```

Return value

cm_ignore returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
cm_req_srv_t  m;

cm_ignore_clear( cm, CM_CB_SERVER, (CS_VOID*)&m );\

/*Ignore all incoming messages from Adaptive Server
  "SYB_ASE!".
```

```
*/
strcpy( (char*)m.cur_server, "SYB_ASE!" );

if (cm_ignore( cm, CM_CB_SERVER, (CS_VOID*)&m )
   != CS_SUCCEED)
{
   cm_error( "Can't ignore msgs from SYB_ASE!\n" );
   return CS_FAIL;
}
```

Usage

- When an Adaptive Server fails, all the connected clients as well as the clients attempting to connect to it receive the same error message. To prevent these similar errors from triggering the failover process multiple times, you can code the CM so it acknowledges only the first lost connection message received and ignores subsequent similar messages on the same server. When the failed server has recovered fully, the CM can unset the previous ignore message so that no messages are ignored.
- The cm_ignore_clear, cm_ignore, and cm_unignore functions cause a CM to automatically discard messages received from OpenSwitch according to a message template.
- The cm_ignore_clear function establishes an empty message template. After it has been used to clear the *msg* structure, the data structure fields that are to be ignored may be set. By passing this populated data structure template to cm_ignore, all future messages matching the template are automatically discarded by the CM until cm_unignore is called with an identical template.
- Messages are ignored only when all fields of the incoming message exactly match all populated fields of the template message. There is currently no facility for providing “or” logic within a single template. This may be achieved only by passing multiple templates to cm_ignore, or by implementing a separate mechanism.

See also

cm_ignore_clear, cm_unignore

cm_ignore_clear

Description

Sets all fields of a message structure to empty values.

Syntax

CS_RETCODE cm_ignore_clear(*cm*, *msg_type*, *msg*)
cm_t **cm*;
CS_INT *msg_type*;
CS_VOID **msg*;

Parameters

cm

A pointer to a CM control structure.

msg_type

the type of message being passed through the *msg* argument. Valid values for *msg_type* are:

Callback type	Description
CM_CB_SERVER	A server-name request message

msg

a pointer to a cm_req_srv_st structure, which is defined as:

```
typedef struct cm_req_srv_st {
    CS_INT     spid; /* OpenSwitch spid number */
    CS_CHAR    username[31]; /* Name of the user */
    CS_CHAR    appname[31]; /* Application they are running */
    CS_CHAR    hostname[31]; /* Host client is running on */
    CS_CHAR    database[31]; /* Host client is running on */
    CS_CHAR    pool[31]; /* Pool the user is in */
    CS_INT     rsn_code; /* Reason for request (see cm_rsn.h) */
    CS_CHAR    rsn_text[256]; /* Description of reason */
    CS_CHAR    cur_server[31]; /* Current server they are using */
    CS_CHAR    nxt_server[31]; /* Server they want to go too */
} cm_req_srv_t;
```

Return value

cm_ignore_clear returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
cm_req_srv_t  m;

cm_ignore_clear( cm, CM_CB_SERVER, (CS_VOID*)&m );

/*
* Ignore all messages coming generated from SQL
* Server "SYB_ASE!".
```

```
*/
strcpy( (char*)m.cur_server, "SYB_ASE!" );

if (cm_ignore( cm, CM_CB_SERVER, (CS_VOID*)&m )
   != CS_SUCCEED)
{
   cm_error( "Can't ignore msgs from SYB_ASE!\n" );
   return CS_FAIL;
}
```

Usage

- When an Adaptive Server fails, all the connected clients as well as the clients attempting to connect to it receive the same error message. To prevent these similar errors from triggering the failover process multiple times, you can code the CM so it acknowledges only the first lost connection message received and ignores subsequent similar messages on the same server. When the failed server has recovered fully, the CM can then unset the previous ignore message so that no messages are ignored.
- The cm_ignore_clear function establishes an empty message template. After you use it to clear the *msg* structure, set the data-structure fields to ignore. By passing this populated data-structure template to cm_ignore, all future messages matching the template are automatically discarded by the CM until cm_unignore is called with an identical template.
- Messages are ignored only when all fields of the incoming message match exactly all populated fields of the template message. There is currently no facility for providing “or” logic within a single template. This may be achieved only by passing multiple templates to cm_ignore, or by implementing a separate mechanism.

See also

cm_ignore, cm_unignore

cm_init

Description

Initializes a CM context.

Syntax

CS_RETCODE cm_init(*cm_ctx*)
cm_ctx_t **cm_ctx*;

Parameters

cm_ctx

The address of a cm_ctx_t pointer. cm_init sets **cm_ctx* to the address of a newly allocated cm_ctx_t structure.

Return value

cm_init returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
cm_ctx_t   *ctx;

if (cm_init( &ctx ) != CS_SUCCEED)
{
   cm_error( "Unable to allocate context\n" );
   return CS_FAIL;
}
```

Usage

- A CM context structure is used to manage zero or more CMs. It provides a handle for manipulating multiple CMs as a single entity. For example, you can use a CM program to manage multiple OpenSwitch servers at the same time. To do this, you must create multiple CMs, each one connecting to a different OpenSwitch. Multiple CMs are particularly useful in a redundancy setup to eliminate the single point of failure that a single OpenSwitch might pose.
- After a CM context structure has been allocated, individual CM managers may be allocated using cm_create.
- A CM context structure may be destroyed using cm_exit.
- Usually, only one CM context exists per executable.
- Common reasons for failure include:
  - Memory allocation failure
  - A problem with the Open Client installation

See also

cm_create, cm_exit

cm_optreset

Description — Resets the state of option parsing for cm_getopt.

Syntax — CS_RETCODE cm_optreset()

Parameters — None.

Return value — cm_optreset returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_optreset())
{
   cm_error( "Cannot reset options\n" );
   return CS_FAIL;
}
```

Usage

- The cm_getopt function is a utility function similar to the standard UNIX libc function call, getopt(3c). Each subsequent call to cm_getopt parses the next command line option.
- cm_getopt and cm_optreset provide a more portable interface than getopt(3c) and are recommended instead.
- Calling cm_optreset resets the state of cm_getopt to start at the beginning of the supplied command line options.

See also — cm_getopt

cm_ping

Description

Verifies the health of a remote server by checking if it responds to a user connection and a simple request.

Syntax

```
CS_RETCODE cm_ping(cm, server, username, password, database)
    cm_t *cm;
    CS_CHAR *server;
    CS_CHAR *username;
    CS_CHAR *password;
    CS_CHAR *database;
```

Parameters

cm

Pointer to a CM control structure.

server

The name of the remote server to ping, as listed in the *interfaces* file on UNIX and in the *sql.ini* file on Windows.

username

The user name used to connect to the remote server to perform the argv. This user name must exist on the remote server and have access to the database specified by the database argument.

password

The user password used to connect to the remote server.

database

If not NULL, the name of the database to ping on the remote server.

Return value

cm_ping returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully. The Adaptive Server was successfully pinged and appears to be available.
CS_FAIL	The routine failed or the Adaptive Server was not available.

Examples

```
if (cm_ping( cm, "SYB_ASE1", "bob", "bobs_password",
   "pubs2" ) != CS_SUCCEED))
{
   cm_error( "Server SYB_ASE1 is dead.\n" );
   return CS_FAIL;
}
```

Usage

- cm_ping is a utility function used to ping a remote server.
- A server is considered to be alive if:

  - A connection is successfully established to server using *username* and *password*, and
  - If the database is not NULL, a use database command succeeds, or
  - If the database is NULL, a *spid* statement succeeds.
- When the network between the CM host and the remote server goes down, cm_ping can take as long as 60 seconds to return a failure. To be notified of the failure sooner than that, use cm_short_ping instead and specify a time-out value you want for your systems.
- To ping the server more than once before taking the necessary failover actions, use cm_repeat_ping or cm_repeat_short_ping. These functions ping the remote server up to the specified number of times before returning a failure.
- Use cm_ping only on Sybase products that support use database and *spid*.

See also cm_repeat_ping, cm_repeat_short_ping, cm_short_ping

cm_ping_enc

Description

Similar to cm_ping, except it allows for the use of encrypted user names and passwords. Calls cm_repeat_short_ping if *maxretry* and *timeout* are greater than zero, cm_repeat_ping if *maxretry* is greater than zero and *timeout* is not, and cm_short_ping if neither *maxretry* or *timeout* are greater than zero.

Syntax

CS_RETCODE CS_PUBLIC cm_ping_enc(*cm, server, username, password, database, timeout, maxretry, waitsec*, encrypted)
cm_t **cm*
CS_CHAR **server*
CS_CHAR **username*
CS_CHAR **password*
CS_CHAR **database*
CS_INT **timeout*
CS_INT **maxretry*
CS_INT **waitsec*
CS_BOOL encrypted

Parameters

cm
A pointer to a CM control structure.

server
The name of the remote server to ping, as listed in the *interfaces* (UNIX) and *sql.ini* (Windows) files.

username
The user name to connect to the remote server to perform the ping.

password
The user password to use to connect to the remote server.

database
If not NULL, the name of the database to ping on the remote server.

timeout
The timeout value in seconds for the CM to connect to the *servername* specified. If the connection is not established within the amount of time specified, this function returns CS_FAIL. Set this value slightly longer than the usual amount of time it takes for the CM host to ping the server host under normal operating conditions.

maxretry
If failure occurs, the number of times the CM retries to check the server health. If the CM succeeds immediately, cm_ping_enc returns immediately without retrying.

waitsec

The duration, in seconds, that the CM should wait between each retry. If the CM succeeds immediately, cm_ping_enc returns without waiting.

encrypted

A Boolean value that defines whether or not the *username* and *password* are encrypted. If encrypted is set to CS_TRUE, all user names and passwords passed to cm_ping_enc must be encrypted. If set to CS_FALSE, none of the user names and passwords should be encrypted.

Return value

cm_ping_enc returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully. The Adaptive Server was successfully pinged and appears to be available.
CS_FAIL	The routine failed or the Adaptive Server was not available.

Examples

```
if (cm_ping_enc( cm, "SYB_ASE1"
  "0x010373d3657426eafbc917cf04a17456e5347612cd91e756c
  8b6afddb0325574",
  "0x010d43e3555092fafc20955d5647496877186a433f006d7e0
  7df70ae39a7cf3b", pubs2", 30, 3, 20,1 ) !=
  CS_SUCCEED))
{
   cm_error( "Server SYB_ASE1 is dead.\n" );
   return CS_FAIL;
}
```

Usage

- Same as cm_repeat_short_ping, with the additional ability to support encrypted user name and password.
- See the “Usage” section for cm_repeat_short_ping.

See also

cm_repeat_ping, cm_short_ping, cm_repeat_short_ping

cm_repeat_ping

Description

Verifies the health of a remote server, repeating up to the specified number of times if a failure is encountered.

Syntax

CS_RETCODE cm_repeat_ping(*cm*, *server*, *username*, *password*, *database*, *maxretry*, *waitsec*)
cm_t **cm*
CS_CHAR **server*
CS_CHAR **username*
CS_CHAR **password*
CS_CHAR **database*
CS_INT **maxretry*
CS_INT **waitsec*

Parameters

cm
Pointer to a CM control structure.

server
The name of the remote server to ping, as listed in the UNIX *interfaces* file or the Windows *sql.ini* file.

username
The user name used to connect to the remote server to perform the ping. This user name must exist on the remote server and have access to the database specified by the database argument.

password
The user password to be used to connect to the remote server.

database
If not NULL, the name of the database to ping on the remote server.

maxretry
If a failure is encountered, the number of times this function retries before returning. If the ping succeeds immediately, cm_repeat_ping returns without retrying.

waitsec
The duration in seconds this function waits between each retry. If the ping succeeds immediately, cm_repeat_ping returns without waiting.

Return value

cm_repeat_ping returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully. The Adaptive Server was successfully pinged and appears to be available.

Return value	Meaning
CS_FAIL	The routine failed or the Adaptive Server was not available.

Examples

```
if (cm_repeat_ping( cm, "SYB_ASE1", "bob", "bobs_password", "pubs2", 3, 5 )
   != CS_SUCCEED))
{
    cm_error( "Failed to connect to SYB_ASE1 after 3  retries.\n" );
    return CS_FAIL;
}
```

Usage

- cm_repeat_ping is a utility function that can ping a remote server. If the ping succeeds, cm_repeat_ping returns immediately. If the ping fails, cm_repeat_ping sleeps for a specified duration (*waitsec*), then tries to ping the server again. This process repeats until the maximum number of retries (*maxretry*) completes.
- A server is considered to be alive if:
  - A connection is successfully established to server using *username* and *password*, and
  - If the database is not NULL, a use database command succeeds, or
  - If the database is NULL, a *spid* statement succeeds.
- You can use cm_repeat_ping only on Sybase products that support use database and *spid*.

See also

cm_ping, cm_short_ping, cm_repeat_short_ping

cm_repeat_short_ping

Description

Similar to cm_repeat_ping, except that cm_repeat_short_ping also sets a time limit on the duration each ping waits when a failure occurs.

Syntax

CS_RETCODE cm_repeat_short_ping(*cm*, *server*, *username*, *password*, *database*, *timeout*, *maxretry*, *waitsec*)
cm_t **cm*
CS_CHAR **server*
CS_CHAR **username*
CS_CHAR **password*
CS_CHAR **database*
CS_INT **timeout*
CS_INT **maxretry*
CS_INT **waitsec*

Parameters

cm
Pointer to a CM control structure.

server
The name of the remote server to ping, as listed in the *interfaces* (UNIX) or *sql.ini* (Windows) file.

username
The user name used to connect to the remote server to perform the ping. This user name must exist on the remote server and have access to the database specified by the database argument.

password
The user password used to connect to the remote server.

database
If not NULL, the name of the database to ping on the remote server.

timeout
The timeout value in seconds for the user to connect to the *servername* specified to determine the health of the server. If the connection fails within the amount of time specified by this value, this function returns CS_FAIL. Set this value to a number slightly longer than the usual amount of time it takes the CM host to ping the host of the server under normal operating conditions.

maxretry
If a failure is encountered, the number of times this function retries before returning. If the ping succeeds immediately, cm_repeat_short_ping returns without retrying.

waitsec

The duration in seconds this function waits between each retry. If the ping succeeds immediately, cm_repeat_short_ping returns without waiting.

Return value

cm_repeat_short_ping returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully. The Adaptive Server was successfully pinged and appears to be available.
CS_FAIL	The routine failed, or the Adaptive Server was not available, or the host of the Adaptive Server was down and inaccessible through the network.

Examples

```
if (cm_repeat_short_ping( cm, "SYB_ASE1", "bob", "bobs_password", "pubs2",
    15, 3, 5 ) != CS_SUCCEED))
{
cm_error( "Failed to access server SYB_ASE1 after 3 tries.\n");
/* Optional: Do further checks to determine the root cause */
sprintf(cmd, "ping server1");
if (system(cmd) != 0)
{
cm_error( "Host of SYB_ASE1 not responding.\n" );
}
return CS_FAIL;
}
```

Usage

- cm_repeat_short_ping is a utility function that can ping a remote server. If the ping succeeds, cm_repeat_short_ping returns immediately. If the ping fails, or a duration of timeout elapses without a response from the remote server, cm_repeat_short_ping sleeps for a specified duration (*waitsec*), then tries to ping the server again. This process repeats until the maximum number of retries (*maxretry*) is carried out.
- A server is considered to be alive if:
  - A connection is successfully established to the server using *username* and *password*, and
  - The database is not NULL, a use database command succeeds, or
  - The database is NULL, a *spid* statement succeeds.
- You can use cm_repeat_short_ping only on Sybase products that support use database and *spid*.

- cm_repeat_short_ping can return false failures if *timeout* is set to a value that is too small, or if the network is sluggish. Sybase recommends that you perform further analysis to determine the precise reason for its failure before triggering a failover.

See also cm_ping, cm_short_ping

cm_run

Description

Starts the CM.

Syntax

CS_RETCODE cm_run(*ctx*)
cm_ctx_t **ctx*;

Parameters

ctx

Pointer to a CM context structure.

Return value

cm_run returns this value:

Return value	Meaning
`CS_FAIL`	cm_run failed or a callback handler returned `CS_FAIL`.

Examples

```
if (cm_run( ctx ) != CS_SUCCEED)
{
   cm_error( "coordination module done.\n" );
   return CS_FAIL;
}
```

Usage

- cm_run acts as the main dispatch loop for the CM. It waits for incoming OpenSwitch events and dispatches them to the appropriate event handler installed with cm_callback.
- cm_run does not exit unless an internal error is encountered, or if a callback handler returns a CS_FAIL.
- cm_run may be called even when no CMs are connected to an OpenSwitch server. In this case, only timed callbacks installed with cm_timer_add are executed.

See also

cm_run, cm_timer_add

cm_set_print

Description

Installs an error display function.

Syntax

CS_RETCODE cm_set_print(*print_func*)
cm_printerr_fn **print_func*;

Parameters

print_func

A NULL, or a pointer to a function of the form:

```
CS_RETCODE print_func( str )
   CS_CHAR *str;
```

Return value

cm_set_print returns these values:

Return Value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
CS_RETCODE print_func( str )
   CS_CHAR *str;
{
   fputs( str, stdout );
   return CS_SUCCEED;
}
...

if (cm_set_print( print_func ) != CS_SUCCEED)
{
   cm_error( "Unable to install print_func\n" );
   return CS_FAIL;
}
```

Usage

- By default, cm_error and all internal error messages display to *stderr*. The cm_set_print function may be used to replace the default display method with a custom function; for example, to write messages to a log file.
- If a NULL print_func is supplied, the default display method is used.

See also

cm_error

cm_set_prop

Description

Sets the *prop* attribute of a coordination module *cm* to *value*. The meaning of *value* depends on which property is being manipulated.

Syntax

CS_RETCODE cm_set_prop(*cm*, *prop*, *value*)
cm_t **cm*;
CS_INT *prop*;
CS_VOID **value*;

Parameters

cm

a pointer to a CM control structure, which is the structure used to represent a CM.

prop

the name of the property to be set. Valid entries for *prop* are:

- CM_P_USERDATA – allows a user-created application to store a value that may be used by the callback function at a later time. Callback routines are asyncronous and are defined using cm_callback as the function to call back for a particular event.
- CM_P_ASYNC – a Boolean value that turns on or off whether notifications are sent directly at the time of receipt.

value

The value to which the specified *prop* is being set.

If *prop* is:	***value* can be:**
CM_P_USERDATA	A pointer to data to be passed to the callback function when executed.
CM_P_ASYNC	• CS_FALSE – to place notifications in a queue to be sent one at a time. This is the default. • CS_TRUE – to send notifications directly at the time of receipt.

Return value

cm_set_prop returns these values:

Return value	**Meaning**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
CS_VOID  *data;

data = (CS_VOID*)strdup( "STRING" );

if (cm_set_prop(cm, CM_P_USERDATA, data )
```

```
        != CS_SUCCEED)
    {
        cm_error("Unable to set USERDATA property\n");
        return CS_FAIL;
    }
```

Usage

Although the contents of all CM data structures are transparent (versus opaque), do not directly access fields within the data structures. Instead, use the cm_get_prop or cm_set_prop routines. This allows the internal definitions to be changed in future releases without affecting existing code.

See also

cm_get_prop

cm_short_ping

Description

Verifies the health of a remote server by checking if it responds to a user connection and a simple request within a specified amount of time.

Syntax

```
CS_RETCODE cm_short_ping(cm, server, username, password,
        database, timeout)
    cm_t *cm
    CS_CHAR *servername
    CS_CHAR *username
    CS_CHAR *password
    CS_CHAR *database
    CS_INT *timeout
```

Parameters

cm

Pointer to a CM control structure.

server

The name of the remote server to ping, as listed in the *interfaces* (UNIX) or *sql.ini* (Windows) file.

username

The user name used to connect to the remote server to perform the ping. This user name must exist on the remote server and have access to the database specified by the database argument.

password

The user password to use to connect to the remote server.

database

If not NULL, the name of the database to ping on the remote server.

timeout

The timeout value in seconds for the user to connect to the *servername* specified to determine the health of the server. If the connection is not established within the amount of time specified, this function returns CS_FAIL. Set this value slightly longer than the usual amount of time it takes for the CM host to ping the server host under normal operating conditions.

Return value

cm_short_ping returns these values:

Return value	Meaning
`CS_SUCCEED`	The routine completed successfully. The Adaptive Server was successfully pinged and appears to be available.

Return value	Meaning
CS_FAIL	The routine failed or the Adaptive Server was not available. Alternatively, the host of the Adaptive Server was not responding or inaccessible through the network.

Examples

```
if (cm_short_ping( cm, "SYB_ASE1", "bob", "bobs_password", "pubs2", 15 )!=
    CS_SUCCEED)
{
       cm_error( "Failed to access server SYB_ASE1 within 15 seconds.\n");
       /* Optional: Do further checks to determine the root cause */
       sprintf(cmd, "ping server1");
       if (system(cmd) != 0)
       {
                cm_error( "Host of SYB_ASE1 not responding.\n" );
       }
       return CS_FAIL;
}
```

Usage

- cm_short_ping can ping a remote server.
- A server is considered to be alive if:
  - A connection is successfully established to the server using *username* and *password*, and
  - If the database is not NULL, a use database command succeeds, or
  - If the database is NULL, a *spid* statement succeeds.
- You can use cm_short_ping only on Sybase products that support use database and *spid*.
- cm_short_ping can return false failures if the timeout value is set too low or if the network is slow. Therefore, Sybase recommends that you perform further analysis to determine the reason for a failure before triggering a failover.

cm_start

Description

Resumes activity of connections.

Syntax

CS_RETCODE cm_start(*cm*, *pool*, *server*, *spid*)
cm_t **cm*;
CS_CHAR *pool*;
CS_CHAR **server*;
CS_INT *spid*;

Parameters

cm

A pointer to a CM control structure.

pool

The name of the pool in which the connections should be started. Supplying only this argument starts all connections within the pool.

server

Resumes connections to the remote server. Supplying only this argument starts all connections to the server.

spid

Starts the connection identified within OpenSwitch by *spid*. If this argument is specified, *pool* and *server* are ignored. A *spid* value of -1 indicates that no specific connection is to be started.

Return value

cm_start returns these values:

Return Value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_start( cm, NULL, "SYB_ASE1", -1)
   != CS_SUCCEED)
{
   cm_error(
   "Can't start connections on SYB_ASE1\n" );
   return CS_FAIL;
}
```

Usage

- cm_start is used to resume connections in OpenSwitch that were previously stopped using cm_stop.
- cm_start is implemented in terms of the rp_start registered procedure within OpenSwitch. For details, see the *OpenSwitch Administration Guide*.
- Passing a NULL value for *pool* or *server* or a value of -1 for *spid* acts as a wildcard for that field, indicating that all client connections match.

- If no arguments are supplied to cm_start, all connections are started within OpenSwitch.
- *spid* refers to the OpenSwitch process ID, not the process ID in the remote Adaptive Server; these two values are not the same.
- Starting a connection that was not stopped has no effect.

See also cm_stop

cm_stop

Description

Suspends connection activity.

Syntax

CS_RETCODE cm_stop(*cm*, *pool*, *server*, *spid*, *flags*)
cm_t **cm*;
CS_CHAR **pool*;
CS_CHAR **server*;
CS_INT *spid*;
CS_INT *flags*;

Parameters

cm

A pointer to a CM control structure.

pool

The name of the pool in which the connections should be stopped. Supplying only this argument stops all connections within the *pool*.

server

Suspends connections to the remote server. Supplying only this argument stops all connections to the server.

spid

Stops the connection identified within the OpenSwitch by *spid*. If this argument is specified, *pool* and *server* are ignored. An *spid* value of -1 indicates that no specific connection is to be stopped.

flags

Symbolic options that indicate how to stop connections. These options may be used with "or" statements. Valid values for *flags* are:

Status	Description
CM_IGNTRAN	Stops connections even if they are in the middle of a transaction. Without this flag, cm_stop waits for the current transaction to complete.
CM_IGNFAIL	Causes stopped connections to ignore the failure of a Adaptive Server; that is, failure messages are broadcast to the CM due to the failure, and a reconnect attempt is made when cm_start is issued.

Return value

cm_stop returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_stop( cm, NULL, "SYB_ASE1", -1, CS_IGNTRAN)
   != CS_SUCCEED)
```

```
{
   cm_error(
   "Can't stop connections on SYB_ASE1\n" );
   return CS_FAIL;
}
```

Usage

- cm_stop is used to suspend connections in OpenSwitch. Connections matching *pool*, *server*, and *spid* are stopped as soon as their transactions have completed (unless the CM_IGNTRAN flag is supplied) and as soon as the currently executing query has completed.
- cm_stop is implemented in terms of the rp_stop registered procedure within OpenSwitch. For more details, see the *OpenSwitch Administration Guide*.
- Passing a NULL value for *pool* or *server* or a value of -1 for *spid* acts as a wildcard for that field, indicating that all client connections match.
- *spid* refers to the OpenSwitch process ID, not the process ID in the remote Adaptive Server; these two values are not the same.
- Stopping a connection that is already stopped has no effect.
- cm_stop applies only to connections that are already established in the OpenSwitch server. It does not apply to connections established after it is called.

See also

cm_start

cm_timer_add

Description

Adds a timed callback.

Syntax

CS_RETCODE cm_timer_add(*cm*, *name*, *ms*, *func*, *data*, *flags*)
cm_t **cm*;
CS_CHAR **name*;
CS_INT *ms*;
cm_timer_cb **func*;
CS_VOID **data*;
CS_INT *flags*;

Parameters

cm
A pointer to a CM control structure.

name
The symbolic name for the callback.

ms
the number of milliseconds until the callback is executed.

func
A pointer to a callback function.

data
A pointer to data to be passed to the callback function when executed.

flags
Flags to affect the manner in which the timer callback is executed. The only valid value for *flags* is:

Status	Description
CM_TF_ONCE	The callback is called only once, at which time it is removed from the list of active callback functions and can be reinstalled only by calling cm_timer_add again.

Return value

cm_timer_add returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
CS_RETCODE cb_func( cm, name, data )
   cm_t     *cm;
   CS_CHAR  *name;
   CS_VOID  *data;
{
   printf( "%s: data is %s\n",
      (char*)name, (char*)data );
```

```
    return CS_SUCCEED;
}

if (cm_timer_add( cm, "Callback #1", (CS_INT)30000,
    (cm_timer_cb*)cb_func,
    (CS_VOID*)NULL, (CS_INT)0)
    != CS_SUCCEED)
{
    cm_error(
    "Unable to install Callback #1\n" );
    return CS_FAIL;
}
```

Usage

- A timed callback is a function that is called automatically every *n* milliseconds by the CM. Timed callbacks supply a mechanism for polling various system resources.
- Timed callbacks are executed synchronously. Therefore, the granularity of the timer varies with the activity of the CM and the number of timer callbacks installed. Do not use timed callbacks where great precision of timing is expected.
- The *name* of the callback is used to determine which callback handler is removed by cm_timer_rem.
- A return value of CS_SUCCEED from a timer callback function indicates that the function completed normally. Returning CS_FAIL causes the CM to exit, and cm_run to return CS_FAIL.

See also

cm_timer_rem, cm_run

cm_timer_rem

Description

Removes a timed callback.

Syntax

CS_RETCODE cm_timer_rem(*cm*, *name*)
cm_t **cm*;
CS_CHAR **name*;

Parameters

cm

A pointer to a CM control structure.

name

The symbolic name for the callback to be removed.

Return value

cm_timer_rem returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_timer_rem( cm, "Callback #1" )
   != CS_SUCCEED)
{
   cm_error(
   "Unable to de-install Callback #1\n" );
   return CS_FAIL;
```

Usage

- A timed callback is a function that is automatically called every *n* milliseconds by the CM.
- Timed callbacks are executed synchronously. Therefore, the granularity of the timer varies with the activity of the CM and the number of timer callbacks installed. Do not use timed callbacks where great precision of timing is expected.
- The *name* of the callback supplied to cm_timer_rem must match the *name* specified for cm_timer_add.

See also

cm_timer_add

cm_unignore

Description

Removes OpenSwitch ignore requests matching template.

Syntax

CS_RETCODE cm_unignore (*cm*, *msg_type*, *msg*)
cm_t **cm*;
CS_INT *msg_type*;
CS_VOID **msg*;

Parameters

cm

Pointer to a CM control structure.

msg_type

The type of message being passed through the *msg* argument. The only valid value for *msg_type* is:

Callback type	Description
CM_CB_SERVER	A server-name request message

msg

A pointer to a valid data structure of the type identified by *msg_type*.

Return value

cm_unignore returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
cm_req_srv_t m;

cm_ignore_clear( cm, CM_CB_SERVER, (CS_VOID*)&m );

/*
* "Unignores" all messages coming generated
* from Adaptive Server "SYB_ASE!".
*/
strcpy( (char*)m.cur_server, "SYB_ASE!" );

if (cm_unignore( cm, CM_CB_SERVER, (CS_VOID*)&m )
   != CS_SUCCEED)
{
   cm_error(
   "Can't unignore msgs from SYB_ASE!\n" );
   return CS_FAIL;
}
```

Usage

- Because one message is received for each OpenSwitch client connection that is lost due to an Adaptive Server failure, you may want to pay attention only to the first message received and, following the failover, ignore any subsequent messages from that Adaptive Server until it is recovered.
- The cm_ignore_clear, cm_ignore, and cm_unignore functions are used to cause a CM to automatically discard messages received from OpenSwitch according to a message template.
- The cm_ignore_clear function establishes an empty message template. After it has been used to clear the *msg* structure, the data structure fields that are to be ignored may be set. By passing this populated data structure template to cm_ignore, all future messages matching the template are automatically discarded by the CM until cm_unignore is called with an identical template.
- Messages are ignored only when all fields of the incoming message match exactly all populated fields of the template message. There is no facility for providing “or” logic within a single template. You can do this only by passing multiple templates to cm_ignore or by implementing a separate mechanism.

See also

cm_ignore_clear, cm_ignore

cm_version

Description	Returns a pointer to the location of the version string and displays the version information for the CM.
Syntax	CS_CHAR * CS_PUBLIC cm_version _ANSI_ARGS((CS_VOID))
Parameters	None.

Examples

```
Sybase Coordination Module/15.0/B/SPARC/Solaris 2.8/0/OPT/Mon Mar 22
12:30:52 2005
Confidential property of Sybase, Inc.
Copyright 1987 - 2005
Sybase, Inc.  All rights reserved.
Unpublished rights reserved under U.S. copyright laws.
This software contains confidential and trade secret information of Sybase,
Inc.  Use, duplication or disclosure of the software and documentation by
the U.S. Government is subject to restrictions set forth in a license
agreement between the Government and Sybase, Inc. or other written
agreement specifying the Government's rights to use the software and any
applicable FAR provisions, for example, FAR 52.227-19.
Sybase, Inc. One Sybase Drive, Dublin, CA 94568, USA
```

Coordination module registered procedures

This section describes registered procedures for OpenSwitch coordination modules (CM). CM registered procedures are issued programatically within the user code to implement registered procedure calls (RPCs) via a CM.

Return values

All cm_rp_* calls return these values:

Value	Description
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Table 3-2: CM registered procedures

Registered procedure	Description
cm_kill	Kills client connections within OpenSwitch.
cm_pool_status	Sets the status of a given pool.
cm_rp_cancel	Cancels the processing of a client connection.
cm_rp_cfg	Reads the OpenSwitch configuration file at runtime.
cm_rp_debug	Enables or disables OpenSwitch debugging messages.
cm_rp_dump	Dumps the thread and/or mutex information.
cm_rp_go	Resumes the activity of the OpenSwitch server after a user has performed some manual intervention.
cm_rp_help	Displays registered procedures and their respective parameters.
cm_rp_msg	Queues text messages to broadcast to one or more client connections.
cm_rp_pool_addattrib	Adds a connection attribute or value to a pool.
cm_rp_pool_addserver	Adds the status of the server within the pool.
cm_rp_pool_cache	Displays or sets the pool cache setting.
cm_rp_pool_create	Creates a new pool.
cm_rp_pool_drop	Drops the existing pool.
cm_rp_pool_help	Displays information about the pools.
cm_rp_pool_remattrib	Removes a connection attribute or value from a pool.
cm_rp_pool_remserver	Removes the server from the pool.
cm_rp_pool_server_status	Displays or sets the status of the server present in the pool.
cm_rp_rcm_connect_primary	Sends a notification to the secondary replication coordination module (RCM) telling it to establish a connection to the primary OpenSwitch.
cm_rp_rcm_list	Displays a list of RCMs with which OpenSwitch is familiar.
cm_rp_rcm_shutdown	Shuts down a given RCM.
cm_rp_rcm_startup	Starts a given RCM.
cm_rp_rmon	Displays the current set of attribute/value pairs being used by the resource governor thread.

Registered procedure	Description
cm_rp_set	Sets or displays a configuration parameter's value.
cm_rp_showquery	Displays a query being executed by the specified *spid*.
cm_rp_shutdown	Shuts down an OpenSwitch server.
cm_rp_version	Displays the version number of OpenSwitch.
cm_rp_who	Displays detailed information about user connections to OpenSwitch.
cm_server_status	Sets the status of a given remote server.
cm_set_srv	Responds to a CM_CB_SERVER message.
cm_switch	Switches connections between servers.

cm_kill

Description

Shuts down client connections within OpenSwitch.

Syntax

CS_RETCODE cm_kill(*cm*, *pool*, *server*, *spid*)
cm_t **cm*;
CS_CHAR **pool*;
CS_CHAR **server*;
CS_INT *spid*;

Parameters

cm

A pointer to a CM control structure.

pool

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

server

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

spid

Shuts down the connection identified within the OpenSwitch by *spid*. If this argument is specified, *pool* and *server* are ignored. An *spid* of -1 indicates that all connections matching the pool name and server name are to be shut down.

Return value

cm_kill returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_kill( cm, NULL, "SYB_ASE1", -1)
   != CS_SUCCEED)
{
   cm_error(
   "Can't kill connections to SYB_ASE1\n" );
   return CS_FAIL;
}
```

Usage

- cm_kill is used to shut down client connections to the remote server through OpenSwitch.
- If no arguments are supplied to cm_kill, all connections are shut down within OpenSwitch. Use this procedure with caution.

- cm_kill is implemented in terms of the rp_kill registered procedure within OpenSwitch. For more details, see the *OpenSwitch Administration Guide*.
- Passing a NULL value for *pool* or *server* or a value of -1 for *spid* acts as a wildcard for that field, indicating that all client connections match.
- *spid* refers to the OpenSwitch process ID, not the process ID in the remote Adaptive Server; these two values are not the same.
- As with Adaptive Server, shutting down a connection causes it to be forcefully removed from the OpenSwitch server, and no messages are delivered to the client.

See also cm_switch, cm_stop, cm_start

cm_pool_status

Description

Sets the status of a given pool.

Syntax

CS_RETCODE cm_pool_status(*cm*, *pool*, *status*)
cm_t **cm*;
CS_CHAR **pool*;
CS_INT *status*;

Parameters

cm

Pointer to a CM control structure.

pool

The name of the pool that is to have its status set.

status –

A symbolic value representing the status to which pool is to be set. Valid values for *status* are:

Status	Description
CM_UP	The pool is immediately available for use.
CM_DOWN	The pool is unavailable, and is not considered for use by any new client connections established to OpenSwitch. New client connections are failed over to the next available pool if one is configured.
CM_LOCKED	The pool is available, but any new incoming connections are blocked (or stopped) until the status is changed to CM_UP or CM_DOWN. But if the *NOWAIT_ON_LOCKED* parameter is set to 1 in the OpenSwitch configuration, clients are rejected immediately, a descriptive message is sent, and blocked connections appear to the client application to have stopped responding until the pool is unlocked.

Return value

cm_pool_status returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_pool_status( cm, "POOLA", "DOWN" )
   != CS_SUCCEED))
{
   cm_error( "Could not mark POOLA as DOWN\n" );
   return CS_FAIL;
 }
```

Usage

- cm_pool_status uses the rp_pool_status registered procedure within OpenSwitch to function. For more details, see the *OpenSwitch Administration Guide*.
- Changing the status of a pool does not affect users who are currently using the pool. The pool status applies only to connections actively being established to OpenSwitch, or existing connections that are in the process of switching or performing a failover.
- Connections that are currently blocked on a locked pool are blocked until either the pool is unlocked or until the client application performs a disconnect. Administrative requests made of the connection, such as a call to cm_switch, or cm_stop, are queued until the pool changes status.
- To stop all activity on a given pool, use cm_pool_status with the CM_LOCKED argument followed by a call to cm_stop.

See also

cm_server_status

cm_rp_cancel

Description

Uses rp_cancel to cancel the processing of a client connection.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_cancel(*cm*, *pool*, *server*, *spid*, *why*)

cm_t **cm*;

CS_CHAR **pool*;

CS_CHAR **server*;

CS_INT *spid*;

CS_CHAR **why*;

Parameters

cm

Pointer to a CM control structure.

pool

Cancels connections to *pool*. Supplying only this argument causes all connections within *pool* to be canceled.

server

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

spid

Cancels the connection identified within the OpenSwitch server by *spid*. If *spid* is -1, connections of all *spid's* connected to *server* in the *pool* are cancelled.

why

Message to be sent to the user of a cancelled query.

Examples

Example 1

```
if (cm_rp_cancel(cm,(char *)NULL, (char *)NULL, -1, "OpenSwitch") !=
     CS_SUCCEED)
{
     cm_error("Unable to cancel all the connections connected to
               the OpenSwitch.\n");
     return  CS_FAIL;
}
```

Cancels all the connections to the OpenSwitch server.

Example 2

```
if (cm_rp_cancel(cm,"POOL1", (char *)NULL, -1, "OpenSwitch") != CS_SUCCEED)
{
     cm_error("Unable to cancel all the connections connected to 'POOL1'
               of the OpenSwitch.\n");
```

```
     return  CS_FAIL;
}
```

Cancels all OpenSwitch connections to “POOL1.”

Example 3

```
if (cm_rp_cancel(cm,”POOL1”,“ASE”,17,"OpenSwitch")  != CS_SUCCEED)
{
     cm_error(“Unable to cancel the connection having spid ‘17’ connected
               to server ‘ASE’ of pool ‘POOL1’ of the OpenSwitch.\n”);
     return  CS_FAIL;
}
```

Cancels a connection where *spid* 17 is connected to server “ASE” in “POOL1.”

cm_rp_cfg

Description

Uses rp_cfg within OpenSwitch to read the OpenSwitch configuration file at runtime.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_cfg (*cm*, *cfg_file*)

cm_t **cm*;

CS_CHAR **cfg_file*;

Parameters

cm

Pointer to a CM control structure.

cfg_file

The name of the configuration file to be read. Passing a file name of NULL, default, or an empty string causes the previously processed configuration file to be read.

Examples

```
if (cm_rp_cfg(cm, "default")!= CS_SUCCEED)
{
     cm_error(“Unable to read configuration File\n”);
     return  CS_FAIL;
}
```

Reads the configuration file at runtime.

cm_rp_debug

Description

Uses rp_debug within a coordination module to enable or disable OpenSwitch debugging messages.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_debug(*cm*, *flags*, *state*)

cm_t **cm*;

CS_CHAR **flags*;

CS_CHAR **state*;

Parameters

cm

pointer to a CM control structure.

flags

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 empty option (“ ”) lists the debugging flags that are currently enabled. The following table shows the valid debugging 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.
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.
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 made available at compile time).

Value	Description
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.
s	Shows access and release of shared and exclusive internal locks (used to prevent concurrent access to internal data structures).
S	Logs SQL statements that are replayed during failover.
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.
x	Displays mutex accesses (more detailed view on shared locks).

state

State of the flags.

Examples

Example 1

```
if (cm_rp_debug(cm,"i", "on") != CS_SUCCEED)
{
     cm_error("Unable to set 'i' debugging options\n");
     return  CS_FAIL;
}
```

Sets the "i" debugging options.

Example 2

```
if (cm_rp_debug(cm,"i", "off");
{
     cm_error("Unable to reset 'i' debugging options\n");
     return  CS_FAIL;
}
```

Resets the "i" debugging options.

cm_rp_dump

Description

Uses rp_dump to dump the thread and/or mutex information.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_dump(*cm*, *what*, *sendtolog*)

cm_t **cm*;

CS_INT *what*;

CS_INT *sendtolog*;

Parameters

cm

Pointer to a CM control structure.

what

Valid values are:

- CM_THREAD – to dump information about all threads
- CM_MUTEX – to dump information about all mutexes
- CM_ALL – to dump information about all OpenSwitch threads and mutexes.

sendtolog

If *sendtolog* is nonzero, the output is directed to the OpenSwitch log; otherwise, the output is directed to the caller.

Examples

```
if (cm_rp_dump(cm, CM_ALL, 0) != CS_SUCCEED)
{
     cm_error("Unable to dump information\n");
     return  CS_FAIL;
}
```

Dumps threads and mutex information.

cm_rp_go

Description

Uses rp_go to resume the activity of the OpenSwitch after a user has performed some manual intervention.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_go(*cm*)

cm_t **cm*;

Parameters

cm

Pointer to a CM control structure.

Examples

```
if (cm_rp_go(cm) != CS_SUCCEED)
```

```
{
     cm_error("Unable to resumes the activity of the OpenSwitch after a user
               has done some manual intervention \n");
     return  CS_FAIL;
}
```

Resumes the activity of the OpenSwitch after a user has performed some manual intervention.

cm_rp_help

Description — Uses rp_help to display registered procedures and their respective parameters.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_help(*cm*)

cm_t **cm*;

Parameters — *cm*

Pointer to a CM control structure.

Examples

```
if (cm_rp_help(cm) != CS_SUCCEED)
{
     cm_error("Unable to display the list of the registered procedures and
               their respective parameters.\n");
     return  CS_FAIL;
}
```

Displays a list of the registered procedures and their parameters.

cm_rp_msg

Description — Uses rp_msg within OpenSwitch to queue text messages to broadcast to one or more client connections.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_msg(*cm*, *pool*, *server*, *spid*, *msg*)

cm_t **cm*;

CS_CHAR **pool*;

CS_CHAR **server*;

CS_INT *spid*;

CS_CHAR **msg*;

Parameters

cm
Pointer to a CM control structure.

pool
The name of the pool to which the message should be delivered. If only *pool* is specified, the message is sent to all connections within the *pool*.

server
Sends the message to current connections to this *server*. If only *server* is specified, the message is sent to all current connections to the *server*.

spid
The OpenSwitch process ID of the client connection to receive the message. If *spid* is -1, the message is sent to all the *spids* connected to *server* in the *pool*.

msg
The text of the message to be delivered.

Examples

Example 1

```
if (cm_rp_msg(cm,(char *)NULL, (char *)NULL, -1, "All connections will shut
down in 5 minutes") != CS_SUCCEED)
{
     cm_error("Unable to send message to all the connections connected
               to the OpenSwitch.\n");
     return  CS_FAIL;
}
```

Sends a message to all the OpenSwitch connections.

Example 2

```
if (cm_rp_msg(cm,"POOL1", (char *)NULL, -1, ""All connections will shut down
in 5 minutes")  != CS_SUCCEED)

     cm_error("Unable to send message to all the connections connected to
               'POOL1' of the OpenSwitch.\n");
     return  CS_FAIL;
}
```

Sends a message to all OpenSwitch connections to "POOL1."

Example 3

```
if (cm_rp_msg(cm,"POOL1", "ASE",17, ""All connections will shut down in
    5 minutes")  != CS_SUCCEED)
{
     cm_error("Unable to send message to the connection having spid '17'
```

```
        connected to server 'ASE' of 'POOL1' of the OpenSwitch.\n");
    return  CS_FAIL;
}
```

Sends a message to the OpenSwitch connection *spid* 17 connected to server "ASE" in "POOL1".

cm_rp_pool_addattrib

Description

Uses rp_pool_addattrib to add a connection attribute or value to a pool.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_pool_addattrib(*cm*, *pool*, *attrib*, *value*)

cm_t **cm*;

CS_CHAR **pool*;

CS_INT *attrib*;

CS_CHAR **value*;

Parameters

cm

Pointer to a CM control structure.

pool

Name of the pool to which attributes are being added.

attrib

Name of the attribute to be added to the pool. The valid values are:

- CM_USERNAME
- CM_APPNAME
- CM_HOSTNAME

value

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

Examples

```
if (cm_rp_pool_addattrib(cm,"POOL1", CM_APPNAME, "isql")!= CS_SUCCEED)
{
     cm_error("Unable to add 'appname' attribute.\n");
     return  CS_FAIL;
}
```

Adds the "appname" attribute with a value of "isql" to "POOL1."

cm_rp_pool_addserver

Description

Uses rp_pool_addserver to add the status of the server within the pool.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_pool_addserver(*cm*, *pool*, *server*, *rel_server*, *status*, *position*)

cm_t **cm*;

CS_CHAR **pool*;

CS_CHAR **server*;

CS_CHAR **rel_server*;

CS_INT *status*;

CS_INT *position*;

Parameters

cm

Pointer to a CM control structure.

pool

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

server

Name of the server to be added.

rel_server

Name of an existing server name within the *pool*, relative to the *server* being added.

status

Status of the server being added. Valid values for *status* are:

- CM_UP
- CM_DOWN
- CM_LOCKED.

position

Position of the *server* relative to *rel_server*. Valid values are:

- CM_HEAD
- CM_BEFORE
- CM_AFTER
- CM_TAIL.

Examples

```
if (cm_rp_pool_addserver(cm,"POOL1",  "ase2","ase1",CM_UP,CM_AFTER) !=
                        CS_SUCCEED)
```

```
{
     cm_error("Unable to add server 'ase2' with status 'UP' after 'ase1' in
               the pool POOL1'\n")
     return  CS_FAIL;
}
```

Adds server "ase2" after "ase1" with an UP status in "POOL1."

cm_rp_pool_cache

Description

Uses rp_pool_cache to display or set the pool cache.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_pool_cache(*cm*, *pool*, *cache*)

cm_ t **cm*;

CS_CHAR **pool*;

CS_INT *cache*;

Parameters

cm

Pointer to a CM control structure.

pool

Name of the pool to be cached.

cache

The number of seconds that connection caches are held in the pool. Setting this to a value to zero (0) disables future connection caching. If this value is set to -1, it displays the cache values for the pools.

Examples

Example 1

```
if (cm_rp_pool_cache(cm, (char *)NULL, 30) != CS_SUCCEED)
{
     cm_error("Unable to set cache value for all the pools\n");
     return  CS_FAIL;
}
```

Sets the *cache* value for all pools.

Example 2

```
if (cm_rp_pool_cache(cm, (char *)NULL, -1) != CS_SUCCEED)
{
     cm_error("Unable to display the cache value for all the pools\n");
     return  CS_FAIL;
}
```

Displays the *cache* values.

cm_rp_pool_create

Description

Uses rp_pool_create to create a new pool.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_pool_create(*cm*, *pool*, *rel_pool*, *position*, *status*, *mode*)

cm_t **cm*;

CS_CHAR **pool*;

CS_CHAR **rel_pool*;

CS_INT *position*;

CS_INT *status*;

CS_INT *mode*;

Parameters

cm

Pointer to a CM control structure.

pool

Name of the pool to be created.

rel_pool

Name of an existing pool, relative to the *pool* being created.

position

Position of *pool* relative to *rel_pool*. The valid values are CM_HEAD, CM_BEFORE, CM_AFTER, and CM_TAIL.

status

The initial status of the *pool*. The valid values are CM_UP, CM_DOWN, or CM_LOCKED.

mode

Mode of the *pool*. The valid values are CM_CHAINED or CM_BALANCED.

Examples

```
if (cm_rp_pool_create(cm,"POOL2","POOL1",CM_BEFORE, CM_UP, CM_CHAINED) !=
                      CS_SUCCEED)
{
     cm_error("Unable to create 'POOl2' before 'POOL1' in CHAINED mode
               having status 'UP'\n");
     return  CS_FAIL;
}
```

Creates “POOL2” before “POOL1” in CHAINED mode with an UP status.

cm_rp_pool_drop

Description — Uses rp_pool_drop to drop the existing pool.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_pool_drop(*cm*, *pool*)

cm_t **cm*;

CS_CHAR **pool*;

Parameters

cm

Pointer to a CM control structure.

pool

Name of the pool to be dropped.

Examples

```
if (cm_rp_pool_drop(cm,"POOL1") != CS_SUCCEED)
{
     cm_error(“Unable drop pool ‘POOL1’ \n”);
     return  CS_FAIL;
}
```

Drops “POOL1.”

cm_rp_pool_help

Description — Uses rp_pool_help to display information about pools.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_pool_help(*cm*, *pool*)

cm_t **cm*;

CS_CHAR **pool*;

Parameters

cm

Pointer to a CM control structure.

pool

The name of the pool that is displaying information.

Examples — **Example 1**

```
if (cm_rp_pool_help(cm,"POOL1")  != CS_SUCCEED)
{
     cm_error("Unable to display information for the pool 'POOL1' \n");
     return  CS_FAIL;
}
```

Displays information about "POOL1."

Example 2

```
if (cm_rp_pool_help(cm, (char *)NULL) != CS_SUCCEED)
{
     cm_error("Unable to display information about all the pools \n");
     return  CS_FAIL;
}
```

Displays information about all pools.

cm_rp_pool_remattrib

Description — Uses rp_pool_remattrib to remove a connection attribute or value from a pool.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_pool_remattrib(*cm*, *pool*, *attrib*, *value*)

cm_t **cm*;

CS_CHAR **pool*;

CS_INT *attrib*;

CS_CHAR **value*;

Parameters

cm
Pointer to a CM control structure.

pool
Name of the pool from which attributes are being removed.

attrib
Name of the attribute to be deleted from the pool. The valid values are CM_USERNAME, CM_APPNAME, or CM_HOSTNAME.

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

Examples

```
if (cm_rp_pool_remattrib(cm,"POOL1", CM_APPNAME, "isql") != CS_SUCCEED)
{
```

```
        cm_error("Unable to remove 'appname' attribute.\n");
        return  CS_FAIL;
    }
```

Removes the "appname" attribute with the value of "isql" from "POOL1."

cm_rp_pool_remserver

Description — Uses rp_pool_remserver to remove the server from the pool.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_pool_remserver(*cm*, *pool*, *server*)

cm_t **cm*;

CS_CHAR **pool*;

CS_CHAR **server*;

Parameters

cm
Pointer to a CM control structure.

pool
Name of the pool from which server is to be removed.

server
Name of the server to be removed.

Examples

```
if (cm_rp_pool_remserver(cm,"POOL1",  "ase2") != CS_SUCCEED)
{
     cm_error("Unable to remove server 'ase2' from the pool 'POOL1'\n");
     return  CS_FAIL;
}
```

Removes server "ase2" from "POOL1."

cm_rp_pool_server_status

Description — Uses rp_pool_server_status to display or set the status of the server present in the pool.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_pool_server_status(*cm*, *pool*, *server*, *status*)

cm_ t *cm;

CS_CHAR *pool;

CS_CHAR *server;

CS_INT status;

Parameters

cm
Pointer to a CM control structure.

pool
The name of the pool.

server
The name of the server. If server name is NULL, then cm_rp_pool_server_status displays the status of all servers present in the pool.

status
The status of the *server*. Valid status values are CM_UP, CM_DOWN, and CM_LOCKED.

Examples

```
if (cm_rp_pool_server_status(cm, "POOL1", "ase1", CM_DOWN) != CS_SUCCEED)
{
     cm_error("Unable to set the status of the server 'ase1' present
               in the pool 'POOL1'.\n");
     return  CS_FAIL;
}
```

Sets the status of server "ase1," which is present in "POOL1," to DOWN.

cm_rp_rcm_connect_primary

Description

Issue rp_rcm_connect_primary through a registered procedure call to a secondary OpenSwitch to send a notification to the secondary RCM telling it that the primary OpenSwitch has restarted and it can re-establish a monitoring connection.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_rcm_connect_primary(*cm*)

cm_t *cm;

Parameters

cm
Pointer to a CM control structure.

Examples

```
if (cm_rp_rcm_connect_primary(cm) != CS_SUCCEED)
{
     cm_error("Unable to send the notification.\n");
     return  CS_FAIL;
}
```

Sends the notification to the secondary RCM.

Usage — Used when the primary OpenSwitch starts after the secondary replication coordination module has already been running.

cm_rp_rcm_list

Description — Uses rp_rcm_list to display a list of RCMs with which OpenSwitch is familiar.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_rcm_list(*cm*)

cm_t **cm*;

Parameters — *cm*

Pointer to a CM control structure.

Examples

```
if (cm_rp_rcm_list(cm) != CS_SUCCEED)
{
     cm_error("Unable to display the RCM list\n");
     return  CS_FAIL;
}
```

Displays the RCM list known to OpenSwitch.

cm_rp_rcm_shutdown

Description — Uses rp_rcm_shutdown to shut down a given RCM.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_rcm_shutdown(*cm*, *rcm_name*)

cm_t **cm*;

CS_CHAR **rcm_name*;

Parameters

cm
Pointer to a CM control structure.

rcm_name
Name of the RCM to be shut down.

Examples

```
if (cm_rp_rcm_shutdown(cm, "rcm1") != CS_SUCCEED)
{
     cm_error("Unable to shutdown the 'rcm1'\n");
     return  CS_FAIL;
}
```

Shuts down "rcm1."

cm_rp_rcm_startup

Description

Uses rp_rcm_startup to start the RCM.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_rcm_startup(*cm*, *rcm_path*, *rcm_cfg*, *rcm_log*, *rcm_retries*, *rcm_redundant*)

cm_t **cm*;

CS_CHAR **rcm_path*;

CS_CHAR **rcm_cfg*;

CS_CHAR **rcm_log*;

CS_INT *rcm_retries*;

CS_INT *rcm_redundant*;

Parameters

cm
Pointer to a CM control structure.

rcm_path
Used to specify the path of the RCM. The default value is *$OPENSWITCH/bin/rcm* on UNIX and *%OPENSWITCH%\bin\rcm* on Windows.

rcm_cfg
Used to specify the path of the RCM configuration file. The default is the *RCM_CFG_FILE* value in the OpenSwitch configuration file.

rcm_log
Used to specify the path of the RCM log file. The default is the *RCM_LOG_FILE* value in the OpenSwitch configuration file.

rcm_retries
Used to specify the number of retry attempts made to start an RCM if the RCM exits for reasons other than a user-requested shutdown. If *rcm_retries* is -1, the default is the *RCM_RETRIES* value in the OpenSwitch configuration file.

rcm_redundant
Used to specify whether the RCM is redundant. If *rcm_redundant* is -1, the default is the *RCM_SECONDARY* value in the OpenSwitch configuration file.

Examples

```
if (cm_rp_rcm_startup(cm, (char *)NULL, (char *)NULL, (char *)NULL, -1,-1)
     != CS_SUCCEED)
{
     cm_error("Unable to start the RCM\n");
     return  CS_FAIL;
}
```

Starts the RCM that is present in *$OPENSWITCH/bin/rcm* (UNIX) or *%OPENSWITCH%\bin\rcm* (Windows).

cm_rp_rmon

Description — Uses rp_rmon within OpenSwitch to display the current set of attribute and value pairs being used by the resource governor thread. See "[LIMIT_RESOURCE]" in Chapter 4, "Using the Configuration File," of the *OpenSwitch Administration Guide* for more information about resource monitoring.

Syntax

CS_RETCODE CS_PUBLIC cm_rp_rmon(*cm*)
cm_t **cm*;

Parameters

cm
Pointer to a CM control structure.

Examples

```
if (cm_rp_rmon(cm) != CS_SUCCEED)
{
     cm_error("Unable to display information about the resource governor
               thread. \n");
     return  CS_FAIL;
}
```

Displays information about the resource governor thread.

cm_rp_set

Description — Uses rp_set to set or display a configuration parameter's value.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_set(*cm*, *parm_name*, *parm_value*)

cm_t **cm*;

CS_CHAR **parm_name*;

CS_CHAR **parm_value*;

Parameters

cm
Pointer to a CM control structure.

parm_name
Name of a configuration variable as listed in the configuration file.

parm_value
Value to which the parameter is to be set. If a NULL *parm_value* is supplied, the value of *parm_name* displays.

Examples — **Example 1**

```
if (cm_rp_set(cm,"TEXTSIZE","104857")  != CS_SUCCEED)
{
     cm_error("Unable to set the 'TEXTSIZE' configuration
               parameter\n");
     return  CS_FAIL;
}
```

Sets the value of the *TEXTSIZE* configuration parameter.

Example 2

```
if (cm_rp_set(cm,"TEXTSIZE",(char*)NULL) != CS_SUCCEED)
{
     cm_error ("Unable to display the value of the 'TEXTSIZE'
                configuration parameter.\n");
     return  CS_FAIL;
}
```

Displays the value of the *TEXTSIZE* configuration parameter.

cm_rp_showquery

Description — Uses rp_showquery within OpenSwitch to display query being executed by a *spid*.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_showquery (*cm*, *spid*)

cm_t **cm*;

RCM_SECONDARY CS_INT *spid*;

Parameters

cm
Pointer to a CM control structure.

spid
The OpenSwitch *spid* executing a query.

Examples

```
if (cm_rp_showquery(cm, 7) != CS_SUCCEED)
{
     cm_error("Unable to display query being executed by a spid '7'\n");
     return  CS_FAIL;
}
```

Displays the query being executed by *spid* 7.

cm_rp_shutdown

Description — Uses rp_shutdown within OpenSwitch to shut down an OpenSwitch server.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_shutdown(*cm*)

cm_t **cm*;

Parameters

cm
Pointer to a CM control structure.

Examples

```
if (cm_rp_shutdown(cm) != CS_SUCCEED)
{
     cm_error("Unable to shutdown the OpenSwitch\n");
     return  CS_FAIL;
}
```

Shuts down the OpenSwitch server.

cm_rp_version

Description — Uses rp_version to display the OpenSwitch version number.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_version(*cm*)

cm_t **cm*;

Parameters — *cm*

Pointer to a CM control structure.

Examples

```
if (cm_rp_version(cm) != CS_SUCCEED)
{
     cm_error("Unable to display version number of the OpenSwitch\n");
     return  CS_FAIL;
}
```

Displays the OpenSwitch version number.

cm_rp_who

Description — Uses rp_who to display detailed information about user connections to OpenSwitch.

Syntax — CS_RETCODE CS_PUBLIC cm_rp_who(*cm*, *spid*)

cm t **cm*;

CS_INT *spid*;

Parameters — *cm*

Pointer to a CM control structure.

spid

The OpenSwitch *spid* value to display. "-1" displays information about all *spids* connected to OpenSwitch.

Examples — **Example 1**

```
if (cm_rp_who(cm, 7) != CS_SUCCEED)
{
     cm_error("Unable to display information about spid '7'.\n");
     return  CS_FAIL;
}
```

Displays information about a specific *spid*; for example, *spid* 7.

Example 2

```
if (cm_rp_who(cm, -1) != CS_SUCCEED)
{
     cm_error("Unable to display information about all the spids.\n");
     return  CS_FAIL;
}
```

Displays information about all *spids* connected to OpenSwitch.

cm_server_status

Description — Sets the status of a given remote server.

Syntax

```
CS_RETCODE cm_server_status(cm, server, status)
    cm_t *cm;
    CS_CHAR *server;
    CS_INT status;
```

Parameters

cm

Pointer to a CM control structure.

server

The name of the server that is to have its status set.

status

A symbolic value representing the status to which the server is to be set. Valid values for *status* are:

Status	Description
CM_UP	The server is immediately available for use.
CM_DOWN	The server is unavailable, and is not to be considered for use by any new client connections established to the OpenSwitch server.
CM_LOCKED	The server is available, but any new, incoming connections through the pool are blocked (or stopped) until the status is changed to CM_UP or CM_DOWN, unless the *NOWAIT_ON_LOCKED* parameter is set to 1in the OpenSwitch configuration, in which case clients are rejected immediately and a descriptive message is sent. Blocked connections appear to the client application to be not responding until the pool is unlocked.

Return value — cm_server_status returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_server_status( cm, "SYB_ASE1", "DOWN" )
   != CS_SUCCEED))
{
   cm_error( "Could not mark SYB_ASE1 as DOWN\n" );
   return CS_FAIL;
}
```

Usage

- cm_server_status uses the rp_server_status registered procedure within OpenSwitch to function. For more details, see the *OpenSwitch Administration Guide*.
- Changing the status of a server does not affect users who are currently using the server. The server status applies only to connections actively being established to OpenSwitch, or to existing connections that are in the process of switching or performing a failover.
- Connections that are currently blocked on a LOCKED server remain blocked until the server is unlocked or until the client application performs a disconnect. This means that any administrative requests made of the connection, such as a call to cm_switch, or cm_stop, are queued until the server changes status.
- To stop all activity on a given server, use cm_server_status with the CM_LOCKED argument followed by a call to cm_stop.

See also

cm_pool_status

cm_set_srv

Description

Sets a remote server name for a client to connect to within OpenSwitch in response to a CM_CB_SERVER message.

Syntax

CS_RETCODE cm_set_srv(*cm*, *spid*, *server*)
cm_t **cm*;
CS_INT *spid*;
CS_CHAR **server*;

Parameters

cm

A pointer to a CM control structure.

spid
The OpenSwitch process ID of the connection to be routed.

server
The name of the server to which the *spid* is to be routed.

Return value

cm_set_srv returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_set_srv( cm, (CS_INT)10, "SYB_ASE1" )
   != CS_SUCCEED)
{
   cm_error("To send spid 10 to SYB_ASE1\n");
   return CS_FAIL;
}
```

Usage

- This function may be used in response to a CM_CB_SERVER request, and is used to respond to the calling *spid* with the name of the server that should be used. Usually, the *spid* that issued the CM_CB_SERVER notification blocks waiting for either this function to respond with the name of the server that it should use, or cm_kill to kill the spid, or cm_switch to switch it to another server.
- cm_set_srv utilizes the rp_set_srv registered procedure to function. For more information, see the *OpenSwitch Administration Guide*.
- Calling cm_set_srv on an *spid* that is not actively waiting for a response from a CM does not return an error, and the call has no effect. cm_switch may be used both to switch connections that are not waiting for a response from the CM and those that are.

See also

cm_callback, cm_switch, cm_kill

cm_switch

Description

Switches connections between servers.

Syntax

CS_RETCODE cm_switch(*cm*, *pool_name*, *src_server*, *spid*, *dst_server*, *grace_period*, force)
cm_t **cm*
CS_CHAR **pool_name*
CS_CHAR **src_server*

CS_INT *spid*
CS_CHAR **dst_server*
CS_INT *grace_period*
CS_BOOL force

Parameters

cm

A pointer to a CM control structure.

pool_name

Switches all connections established through *pool_name* to the server specified by *dst_server*. If this parameter is NULL, all pools are assumed.

src_server

Switches all connections currently established to the remote server *src_server* to *dst_server*. If this parameter is NULL, all servers are assumed.

spid

Switches the named OpenSwitch spid to the remote server *dst_server*. If this parameter is -1, all *spids* are assumed.

dst_server

The name of the remote server to which all connections identified by *pool_name*, *src_server*, and *spid* should be switched. If this parameter is NULL, or has a blank value, the connections are switched to the next server as identified by their associated pool.

grace_period

The maximum number of seconds that rp_switch should wait before forcefully switching busy connections. A value of 0 (seconds) indicates that no grace period is to be granted.

force

whether to force connections to switch, even if they are currently busy (either actively communicating with a remote server, or in the middle of an open transaction). Values are:

- CS_TRUE – to force connection switching.
- CS_FALSE – to not force connection switching.

Return value

cm_switch returns these values:

Return value	Meaning
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
if (cm_switch( cm, NULL, "SYB_ASE1", -1,
   "SYB_ASE2", 0, 1)
   != CS_SUCCEED)
```

```
{
   cm_error(
   "Can't switch from SYB_ASE1 to SYB_ASE2\n" );
   return CS_FAIL;
}
```

Usage

- cm_switch uses the OpenSwitch registered procedure rp_switch. For details, see the *OpenSwitch Administration Guide*.
- A call to cm_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 these conditions:
  a If the connection is completely idle (is not communicating with a remote server and is not involved in an open transaction), the connection is silently switched immediately
  b If the connection is busy (either communicating with a remote server or involved in an open transaction), *grace_period* is 0, and *force* is 0, the connection switches as soon as it becomes idle.
  c If the connection is busy, *grace_period* is a positive value, and force is 0, the connection switches as soon as it becomes idle. Otherwise, if *grace_period* seconds pass before it becomes idle, its current query is canceled, and a "deadlock" message is issued to the client. The connection is then switched.
  d If the connection is busy and force is 1, the connection immediately has its query canceled, and receives a "deadlock" message. The connection is then switched.
- The validity of *dst_server* is not checked. Passing an invalid value, for example, an Adaptive Server name that does not exist, causes all incoming client connections to be lost. Use caution when specifying this parameter.
- *dst_server* does not need to be a server within the pool of a given connection, or a server within any pool. It must be a valid server.
- If *force* is 1, then *grace_period* must be zero (0), because *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 cm_stop, a locked pool, or a locked server is processed as soon as the connection becomes unblocked. Forcing a switch has no effect on a blocked connection until it becomes unblocked.

See also

cm_start, cm_stop

CHAPTER 4

Using the Replication Coordination Module

This chapter describes the replication coordination module (RCM), an OpenSwitch sample created using CM APIs, which coordinates failover of a high-availability, warm-standby system.

For information about setting up high-availability, warm-standby environments in Replication Server and Adaptive Server Enterprise, see:

- *Replication Server Administration Guide, Volume 2*
- *Using Sybase Failover in a High Availability Environment* in Adaptive Server Enterprise 15.0 documentation

Introduction

When you install OpenSwitch 15.0, the RCM is installed automatically into the *%OPENSWITCH%* (Windows) or *$OPENSWITCH* (UNIX) directory.

These RCM files are installed in *%OPENSWITCH%\bin* on Windows and in *$OPENSWITCH/bin* on UNIX:

- *rcm.exe* – the replication coordination module executable.
- *runrcm.sh* – a script for starting the RCM.

These files are installed in *%OPENSWITCH%\config* on Windows and in *$OPENSWITCH/config* on UNIX:

- *rcm.cfg* – a sample RCM configuration file.
- *rcm_oswitch.cfg* – a sample OpenSwitch configuration file matching the *rcm.cfg* file.

The *rcm.loc* file, the locales file for the RCM that also contains error messages, is installed in *%OPENSWITCH%\locales* on Windows and in *$OPENSWITCH/locales* on UNIX.

Note Sybase strongly recommends that the RCM and the OpenSwitch server execute on the same machine.

What is the replication coordination module?

The RCM is an OpenSwitch component that coordinates the failover of a high-availability, warm-standby environment.

Note The term "failover" in this document refers to automatically switching to a redundant or standby server when the currently-active server fails or terminates abnormally. It does not refer to Sybase Failover, which is a specific feature of Adaptive Server Enterprise.

A high-availability, warm-standby environment minimally consists of:

- A Replication Server configured for warm-standby replication
- Two Adaptive Server Enterprise servers and corresponding databases
- One OpenSwitch server
- One RCM instance, configured to coordinate failover through the OpenSwitch server

A redundant high-availability, warm-standby environment includes a backup and secondary OpenSwitch, and a backup and redundant RCM. A redundant system minimally consists of:

- A Replication Server configured for warm-standby replication
- Two Adaptive Server Enterprise servers and corresponding databases
- Two OpenSwitch servers

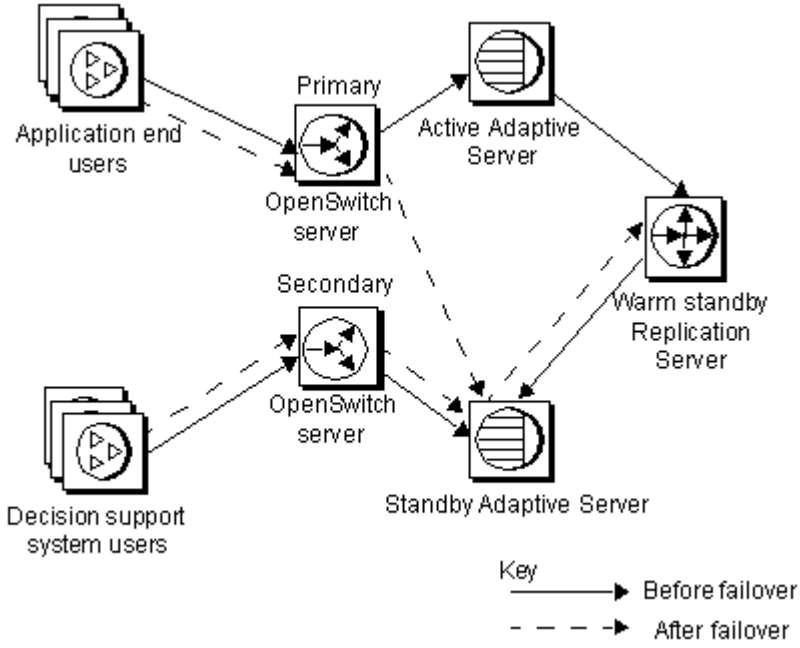
- Two RCM instances configured to coordinate failover through the OpenSwitch servers

Note The RCM does not support concurrent coordination modules. When the RCM establishes a connection to OpenSwitch, OpenSwitch sets the COORD_TIMEOUT to zero (0), which turns off the coordinated CM functionality.

Figure 4-1 illustrates a redundant system before and after the failover of the active Adaptive Server. Before a failover, application end users connect to the active Adaptive Server through the primary OpenSwitch server, and decision-support-system users connect to the standby Adaptive Server through either the primary or the secondary OpenSwitch server.

After failover, the primary OpenSwitch server switches the application end users to the standby Adaptive Server. The application end users are still connected through the primary OpenSwitch server, but now are connected to the standby Adaptive Server. Decision-support-system users continue to connect to the standby Adaptive Server through either the primary or the secondary OpenSwitch server.

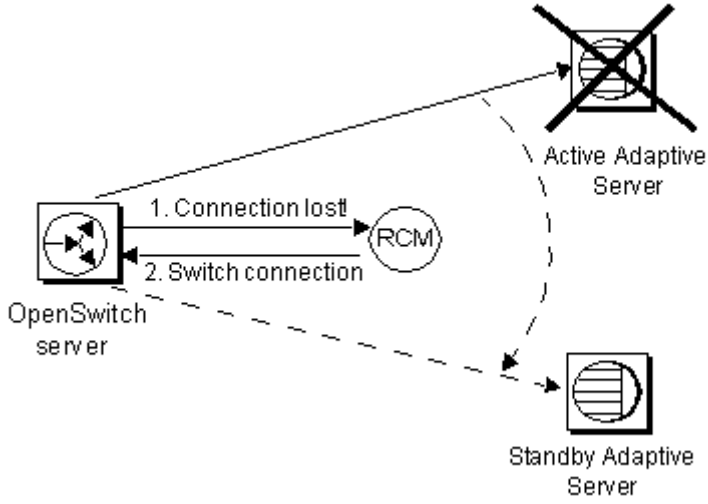
Figure 4-1: A redundant, high-availability, warm-standby environment



CMs coordinate end-user connections that pass through the OpenSwitch server to the Adaptive Servers. If the RCM determines that the active Adaptive Server has failed, it connects to the Replication Server to fail over to the warm-standby server, and coordinates the switch of end users through an OpenSwitch server. Decision-support-system users stay connected through the secondary OpenSwitch server.

Figure 4-2 illustrates the relationship between OpenSwitch and the RCM.

Figure 4-2: How OpenSwitch and the RCM work together



The RCM, in conjunction with Replication Server and OpenSwitch, is designed to help meet the requirements of a high-availability, warm-standby environment. It can:

- Remove all single points of failure
- Achieve an environment with fault-tolerant, redundant servers
- Perform automatic failover of end users when the active data server fails
- Coordinate access to the active and standby Adaptive Servers
- Coordinate a geographically dispersed system, where the active and standby Adaptive Servers are at separate locations

Configuring OpenSwitch and the RCM

The most complicated part of any replication environment is setup and configuration, because there are multiple, interacting servers. This section describes the configuration for each component in the high-availability environment and discusses options for your failover strategy.

Determining your failover strategy

The failover strategies that the RCM supports are: manual, automatic (switch active), and failover with Replication Server quiesce.

- Manual – the RCM notifies the system administrator of a failure, but does not control processing at the Replication Server. The administrator must control the Replication Server.
- Automatic – the RCM coordinates failover automatically and the direction of replication is reversed. This means that the roles of the standby and the active databases have switched. The standby database is now the active database, and changes are replicated from it to what was originally the active database and is now the standby database.
- Failover with Replication Server quiesce – the RCM coordinates failover automatically. The Replication Server is quiesced and replication is stopped. When the Replication Server is quiesced, it does not capture changes in the standby database.

Manual failover

Choose manual failover to control every step of a failure and manually rebuild the active Adaptive Server rather than have transactions automatically applied to the standby database after failover.

Automatic failover or failover with Replication Server quiesce

Choosing between automatic failover and failover with Replication Server quiesce depends on the volume of data coming through the system, the size of the Replication Server queues, and the length of time you expect the active server to be unavailable.

If you choose automatic failover and switch the warm-standby connection, the Replication Server captures all transactions entered at the standby Adaptive Server and stores them so that they can be applied to the active Adaptive Server when the server recovers. This simplifies the recovery process because the active databases do not have to be reloaded. However, if there is a heavy volume of data or the active Adaptive Server is down for an extended period of time, the Replication Server's queue might not be capable of storing all of the transactions. If this is the case, choose failover with Replication Server quiesce.

Understanding a redundant environment

Using two OpenSwitch servers in a high-availability warm-standby environment provides additional robustness and a redundant environment. A redundant environment is one in which there is no single point of failure. The second OpenSwitch server protects the replication environment from the failure of the first OpenSwitch server by assuming control of the failover sequence in case the subsequent active database fails or Replication Server fails. See “Creating a redundant environment” on page 129 for more information.

The second OpenSwitch server can provide secondary access to the active and standby databases. This enables you to load-balance the databases. To load-balance means using all servers simultaneously in the environment until a server fails. You can differentiate user access for each of the databases so that not all users connect to the same database and, thereby, balance the user connection load among databases.

See “DSS users” on page 119 for more information about load balancing.

Planning for high availability

There are several important factors in planning for high availability that can affect your environment setup. These factors are the characteristics of:

- The client application
- The servers
- Replication Server
- OpenSwitch

Coordinating the client application

The important aspects of using client applications with the RCM and OpenSwitch are:

- End-user connectivity
- OpenSwitch restrictions
- Replication Server restrictions

End-user connectivity

The most important reason to coordinate your client application with a high-availability environment is end-user connectivity. Client applications are typically used by two groups of users—application end users, who write to the back-end database, and decision-support-system (DSS) users, who read the data in the database but never write to it. For the purpose of using the RCM and OpenSwitch, keep these two types of users separate.

To keep these two types of users separate, OpenSwitch requires that you divide them into two pools. Pools are groups of user IDs that log in to OpenSwitch to access the back-end database. Each pool has access to a group of servers that you define in the OpenSwitch configuration file.

The RCM tracks the application end-user pool to detect a login failure but ignores the DSS user pool. OpenSwitch tracks the application end-user pool and the DSS user pool; so if a connection switch is required, it can switch each pool to the appropriate database.

See “Configuring user pools” on page 118 for more information.

Before configuring the OpenSwitch server, you must:

1 Identify the users who need continual access to the back-end database.

2 Divide the group of users into application end-user and DSS-user groups.

3 Define one or more user pools.

You must define one user pool for application end users. If you have DSS users in your environment, you can define one or more pools through which they can access the system. If you do not need DSS users to access your system, you do not need a DSS user pool.

Note All application end users must belong to a single user pool.

4 Place the user IDs into their corresponding user pools.

See “Configuring OpenSwitch” on page 116 for more information.

Using client applications with OpenSwitch

OpenSwitch has several inherent restrictions that can affect your application environment:

- Connection context – OpenSwitch does not track and restore current character set, language context, or Adaptive Server context information for a given connection. If the context is changed following the initial connection, the context is lost.
- Performance – when you establish a connection through OpenSwitch, the connect time is doubled, because the connection must first be established to the OpenSwitch, which in turn establishes a connection to the remote Adaptive Server.

  Also, OpenSwitch must closely monitor all traffic passing between the Adaptive Server and the client connection to detect connection context information. This monitoring activity can have an impact on performance, especially when large result sets are involved.
- Number of user connections – because OpenSwitch runs as a single process, the host environment operating system applies constraints on the number of open files per user process.

See the *OpenSwitch Administration Guide* for more information about OpenSwitch restrictions.

Using client applications with Replication Server

Replication Server does not guarantee the replication of cross-database transactions, which are transactions that modify tables in two or more databases on the same server. Replication Server provides transactional integrity within a single database and across the active and standby databases—but not between two databases on the same server.

See the *Replication Server Design Guide* for more information.

Identifying server information required for configuration

To configure your environment, you must gather information about the servers in your environment.

See “Using RCM configuration parameters” on page 116 and “Understanding RCM configuration parameters” on page 122 for more information about configuring servers with OpenSwitch and the RCM, respectively.

Identifying the Adaptive Server Enterprise server pair

The RCM must know the following about the Adaptive Servers in your environment:

- The names of the active and the standby databases and servers as defined in the Sybase *sql.ini* (Windows) or *interfaces* (UNIX) file
- The names of the computers that host the active and standby Adaptive Servers
- The login that has permission to start the Replication Agent thread
- The administrative login the RCM must use

Note This login must be the same on both the active and standby Adaptive Servers.

Identifying the Replication Server

The RCM must know the following about the Replication Server in your environment:

- The name of the Replication Server
- The name of the computer that hosts the Replication Server
- The logical connection names
- The RCM login to Replication Server (set by the *RS_USER* parameter in the RCM configuration file) that has privileges to execute the following commands: switch active, suspend log transfer from all, admin quiesce_force_rsi, admin logical_status, and admin health.

Identifying the OpenSwitch Server

Gather information about your OpenSwitch servers:

- The names of the OpenSwitch servers
- The coordination module user login that the RCM uses
- The active and standby server names
- Configuration settings
- Pool settings

See the *OpenSwitch Administration Guide* for more information.

Identifying pool settings

You must configure OpenSwitch to direct all application end-user pool connections to the active Adaptive Server unless it is down, in which case, to direct them to the standby Adaptive Server.

You can configure the DSS user pool to connect to either of the Adaptive Servers, or to connect only to the standby Adaptive Server.

See “End-user connectivity” on page 112 and “Configuring user pools” on page 118 for more information.

Understanding Replication Server restrictions

A Replication Server supporting a high-availability, warm-standby environment has these restrictions:

- The replicate server cannot be a replicate Replication Server. No other Replication Server can replicate data into the warm-standby Replication Server.
- The replicate server can be a primary Replication Server. Data can be replicated out of a primary Replication Server to a replicate database.

See “Managing a Warm Standby Environment” in the *Replication Server Administration Guide, Volume 2*.

Before configuring the RCM

Before you configure the RCM, you must:

- Identify the logical flow for an automatic failover situation and how the RCM will coordinate this flow through OpenSwitch.
- Identify the likely failover scenarios.
- Identify the server user logins and permissions.
- Identify the names and locations of all servers involved. See “Planning for high availability” on page 111.

Configuring OpenSwitch

This section describes OpenSwitch cofiguration parameters specific to using RCMs. The format of the OpenSwitch configuration file is described in more detail in the *OpenSwitch Administration Guide*, Chapter 4, "Using the Configuration File."

Using RCM configuration parameters

To use an RCM, you must configure OpenSwitch by setting the *COORD_MODE* parameter to ALWAYS. The RCM can then coordinate the switch of users between the active and the standby Adaptive Servers so that the OpenSwitch server does not allow users to connect unless the RCM is available. OpenSwitch determines which server the users are connected to when failover occurs, while the RCM determines the state of each server (either UP or DOWN). If the RCM determines that the active server is down, OpenSwitch switches clients from that server to the standby server.

The parameters in Table 4-1, which are located in the [CONFIG] section of the OpenSwitch configuration file, are vital to the success of coordinated failover, and you must set them correctly.

Table 4-1: Coordinated failover configuration parameters

Parameter	Description	Value
SERVER_NAME	The name of the OpenSwitch server. This parameter must match the RCM *OPENSWITCH* configuration parameter.	OpenSwitch server name
COORD_USER	The user name that the RCM uses to log in to the OpenSwitch server. This parameter must match the RCM *COORD_USER* configuration parameter.	OpenSwitch administrator user name
COORD_PASSWORD	The password that the RCM uses to log in to the OpenSwitch server. This parameter must match the RCM *COORD_PASSWORD* configuration parameter.	OpenSwitch administrator password
COORD_MODE	Setting this parameter to ALWAYS indicates that an RCM is required. For warm-standby environments with coordinated failover, this parameter must be set to ALWAYS.	`ALWAYS`

Configuring RCM autostart

The parameters in Table 4-2, which are also located in the [CONFIG] section of the OpenSwitch configuration file, are used to configure an RCM to automatically start and stop when OpenSwitch starts and stops.

Note The description in Table 4-2 also indicates whether an option is configured dynamically or statically. A dynamic option indicates a newly configured value that takes effect immediately and affects all future connections; existing connections are not affected. Dynamically configured options usually affect individual connections. Static options cannot be changed by the user while OpenSwitch is running. You must stop and restart OpenSwitch before the changes take effect. Static options usually define the overall characteristics of the OpenSwitch server and its start-up options.

See "Starting and stopping the RCM automatically from OpenSwitch" on page 146 and the *OpenSwitch Administration Guide*, Chapter 4, "Starting and Stopping OpenSwitch and RCMs," for complete instructions on configuring this functionality.

Table 4-2: RCM autostart configuration parameters

Parameter	Description	Value
RCM_AUTOSTART	Instruct OpenSwitch whether to start the replication coordination module (RCM). This option is configured dynamically.	Enter: • 0 – to not automatically start the RCM when OpenSwitch starts. This is the default value. • 1 – to automatically start the RCM when you start OpenSwitch.
RCM_RETRIES	The number of times OpenSwitch should retry starting the RCM. If the RCM fails for reasons other than the user requesting that the RCM be shutdown, OpenSwitch attempts to restart the RCM. If an unrequested shut down of the RCM occurs within one minute of starting, OpenSwitch logs an error and does not attempt to restart the RCM. This option is configured statically.	Enter: • 0 – OpenSwitch does not attempt to restart the RCM. • Any numeric value – enter the number of times OpenSwitch should retry to start the RCM.

Parameter	Description	Value
RCM_PATH	The path where OpenSwitch should look for the RCM executable. If you do not enter this path, and are using and 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, and is configured statically.	RCM executable file path
RCM_CFG_FILE	The path where the RCM configuration file is located. This parameter has a NULL value if you do not specify a path, and is configured statically.	RCM configuration file path
RCM_LOG_FILE	The path where the RCM log file should be created. This parameter has a NULL value if you do not specify a path, and is configured statically.	RCM log file path
RCM_SECONDARY	Indicate to OpenSwitch whether the RCM it is launching is a primary or a secondary RCM. The default is "1". This parameter is configured dynamically.	Enter: • 0 – the primary RCM. • 1 – secondary RCM.

Configuring user pools

You have many choices for user connection handling through OpenSwitch; however, you must configure OpenSwitch to have one pool for application end users for use with the RCM. Sybase recommends that you also configure OpenSwitch to have one or more DSS user pools.

The RCM expects to find all application end users in one pool defined in the OpenSwitch configuration file. You can also define and configure one or more user pools for DSS users so that OpenSwitch connects all DSS users to the standby server, and so that the RCM ignores any connection errors they might generate. In a high-performance environment, offloading decision-support-system users to the standby Adaptive Server can minimize performance impact on the active server.

See "End-user connectivity" on page 112 for more information about user pools.

See "DSS users" on page 119 for more information about load balancing.

Application end users

When an application end user logs in, OpenSwitch sends the login request to the RCM. The RCM determines if the user can log in to the requested server based on the state of the replication environment.

- If the environment is active, the user is connected to the active server. If the active server is unavailable, the RCM starts the failover process. (See "Failover processing" on page 156.)
- If the environment has failed over, the user is connected to the standby server. If the standby server is unavailable, the RCM rejects the request, and OpenSwitch notifies the user that the server is down.
- If the environment is in the process of failing over, the request is suspended until the failover is complete. At that time, the user is connected to the standby server.

DSS users

If DSS users log in after the environment has failed over to standby, the RCM either allows the DSS users to access the standby server or rejects them, depending on how you configure OpenSwitch and the RCM.

Other pools can be configured for DSS users. You have more flexibility when setting up this pool because DSS users have read-only access to the Adaptive Servers. The pool can be set to load-balance between servers or set to switch users if a server fails. At that time, all the connections on the failed server are redistributed to the next available server.

See "Setting configuration parameters for user pools" on page 119 for more information.

Setting configuration parameters for user pools

Table 4-3 lists OpenSwitch parameters for user pools. These parameters are in the [CONFIG] section of the OpenSwitch configuration file.

Table 4-3: OpenSwitch user pool configuration parameters

Item	Description	To configure for application end user	To configure for DSS user
POOL	The configuration parameter that defines the name of the user pool.	Set to match the RCM configuration parameter *APP_POOL*.	Set to any string valid for your environment, as long as it is unique.

Item	Description	To configure for application end user	To configure for DSS user
MODE	An argument for the *POOL* parameter that defines the connection mode this user pool uses during failover.	Set to CHAINED. In CHAINED mode, all connections are routed to the first server within the pool. If the first server is not available, the OpenSwitch connects everyone to the next server in the list.	Set to CHAINED or BALANCED. In BALANCED mode, incoming connections are routed among all servers within the pool that have a status of UP. See the *OpenSwitch Administration Guide* for more information.
STATUS	An argument for the *POOL* parameter that defines the status of each server in the pool.	Set to UP as the initial status. The RCM controls the status of each server individually. The RCM monitors the connection and is aware of any failure. If a failure occurs, the RCM changes *STATUS* to DOWN. **Note** If you do not set *STATUS* to UP, RCM does not work properly.	Set to UP as the initial status. The RCM controls the status of each server individually.
SERVER	The configuration parameter that identifies the names of servers in the failover environment.	List the servers in the order they will be used by application end users. List the active server first, followed by the standby server.	List the servers for the DSS users. List the servers in the order they will be used by the DSS users if *MODE* is set to CHAINED.

Item	Description	To configure for application end user	To configure for DSS user
connections	An option for the *POOL* parameter that identifies the user connections used by that pool.	List the user connections that will use the pool defined by the *POOL* parameter. You must list the connections using the following syntax: attribute:regex [, regex] [attribute:regex [, regex]...]] where *attribute* is the name of a connection attribute, such as a user name, an application name, a host name, or a type of connection, and *regex* is a standard SQL-style extended regular expression that describes values for a given attribute. See the *OpenSwitch Administration Guide*, Chapter 4, "Using the Configuration File" for more information. For example, if you set the attribute to "user name", set the regular expression to one of the user names in that pool.	Same as for application end users.

User pool configuration file example

This section shows part of a sample OpenSwitch configuration file that contains a pool for application end users and one for DSS users. The application end-user pool is set up so that application end users connect to the active Adaptive Server first. If it fails, users are switched to the standby Adaptive Server.

The DSS pool is set up so that DSS users connect to the standby Adaptive Server first. If it fails, the users are switched to the active Adaptive Server.

```
[CONFIG]

SERVER_NAME     =  ws_os
CHARSET         =  iso_1
.
.
.
COORD_USER      =  os_coord
COORD_PASSWORD  =  os_coord_pwd
```

```
COORD_MODE     =  ALWAYS
.
.
.
[POOL=Application:MODE=CHAINED, STATUS=UP]
   servers:
      BookServer
      StandbyBook
   connections:
      username:bob
      username:fred


[POOL=DSS:MODE=CHAINED, STATUS=UP]
   servers:
      StandbyBook
      BookServer
   connections:
      username:alice
```

Configuring the RCM

The information the RCM requires to connect to servers in the replication environment is stored in a RCM-specific configuration file, which is in the same location as the OpenSwitch configuration file (*$OPENSWITCH/config* on UNIX and *%OPENSWITCH%\config* on Windows). Because the RCM reads the configuration file only at start-up, you cannot change parameters after the RCM is started. You must restart the RCM to change parameters.

See “Introduction” on page 105 for a list of RCM-specific configuration files.

Understanding RCM configuration parameters

The RCM configuration parameters are set in an RCM-specific configuration file. The configuration file is composed of pairs of parameters and values in the format:

```
parameters=value
```

where *parameter* is the parameter name, and *value* is the value the parameter will be set to when the RCM starts up.

Note Secure the RCM configuration file because it contains passwords for Adaptive Servers, OpenSwitch servers, and Replication Server. To secure the RCM configuration file, set the read and write permissions on the file and the directory.

Table 4-4 lists valid configuration parameters and default values.

Table 4-4: RCM configuration parameters

Parameter	Description	Example	Default
LANGUAGE	The language the RCM uses to communicate with the servers in the replication environment. The RCM also displays error messages in this language.	`japanese`	`us_english`
CHARSET	The character set the RCM uses to communicate with the servers in the replication environment. The RCM also displays error messages using this character set.	`sjis`	`iso_1`
OPENSWITCH	The name of the OpenSwitch associated with the RCM. This parameter must match the OpenSwitch configuration parameter *SERVER_NAME*. This is a required parameter.	`ws_os`	None
COORD_USER	The user name that the RCM uses to connect to the OpenSwitch. The parameter must match the OpenSwitch configuration parameter *COORD_USER*. This is a required parameter.	`os_coord`	None
COORD_PASSWORD	The password that the RCM uses to connect to the OpenSwitch. The parameter must match the OpenSwitch configuration parameter *COORD_PASSWORD*.	`os_coord_pwd`	Empty string

Parameter	Description	Example	Default
RCMNAME	A unique name for an RCM 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 list out to the client application when rp_rcm_list is issued. This parameter is used to support starting an RCM automatically after OpenSwitch starts. See "Starting and stopping the RCM automatically from OpenSwitch" on page 146.	`rcm1_rcm`	The name of the OpenSwitch server specified in the RCM configuration file and appended with "_rcm"; for example: `OSW1_rcm`
REP_SERVER	The name of the Replication Server that controls the warm-standby environment. This is a required parameter.	`ws_rs`	None
RS_USER	The user name that the RCM uses to connect to the Replication Server. The user must have privileges to execute the following commands: switch active, suspend log transfer from all, admin quiesce_force_rsi, admin logical_status, and admin_health. This is a required parameter.	`sa`	None
RS_PASSWORD	The password that the RCM uses to connect to the Replication Server.	`sa_pwd`	Empty string
ACTIVE_ASE	The name of the active Adaptive Server. This is a required parameter.	`BookServer`	None
ACTIVE_USER	The user name that the RCM uses to connect to the active Adaptive Server. The login must have privilege to execute the use database command on all databases defined by the *ACTIVE_DBS* parameter. This parameter is required.	`sa`	None
ACTIVE_PASSWORD	The password that the RCM uses to connect to the active Adaptive Server.	`sa_pwd`	Empty string

Parameter	Description	Example	Default
ACTIVE_DBS	A comma-separated list of databases in the active Adaptive Server that the Replication Server switches to during a failover. The list is used only if the *RS_FAILOVER_MODE* parameter is set to SWITCH. If you do not provide a list, the RCM uses the database names from the logical connection list as the default.	`pubs3`	The default is the list of databases taken from the *LOGICAL_CONN* parameter.
STANDBY_ASE	The name of the standby Adaptive Server. This is a required parameter. In switch active mode only, the standby server must be identified in the Replication Server logical connection definition. See "Managing Warm Standby Applications in the *Replication Server Administration Guide, Volume 2* for more information.	`StandbyBook`	None
STANDBY_USER	The user name that the RCM uses to connect to the standby Adaptive Server. The login must have privileges to execute the sp_start_rep_agent and the use database commands on all databases defined by the *STANDBY_DBS* parameter. This parameter is required.	`sa`	None
STANDBY_PASSWORD	The password that the RCM uses to connect to the standby Adaptive Server.	`sa_pwd`	Empty string
STANDBY_DBS	A comma-separated list of databases in the standby Adaptive Server that the Replication Server switches to during a failover. The list is used only if the *RS_FAILOVER_MODE* parameter is set to SWITCH. If you do not provide a list, the RCM uses the database names from the logical connection list as the default.	`pubs3`	The default is the list of databases taken from the *ACTIVE_DBS* parameter.

Parameter	Description	Example	Default
LOGICAL_CONN	A comma-separated list of Replication Server logical connections in the form *dataserver.database*. This is a required parameter if you have set the *RS_FAILOVER_MODE* parameter to SWITCH.	`LDS.LDB`	None
REQUIRED_DBS	A comma-separated list of databases in the active Adaptive Server that require failover support and that the RCM should ping to determine server failure. If you do not provide this list, the RCM pings only the active Adaptive Server when determining server failure.	`pubs3`	Empty list
APP_POOL	The name of the OpenSwitch pool that identifies all of the application end users. This is a required parameter.	`Application`	None
FAILOVER_WAIT	The number of seconds the RCM waits after a potential failover is detected before initiating the failover process. This failover waiting period gives the active Adaptive Server an opportunity to recover automatically.	`120`	`60`
MONITOR_WAIT	The number of seconds the RCM monitors the Replication Server after invoking a Replication Server failover command (either switch active, or suspend log transfer) and before switching end users to the standby Adaptive Server. This gives the Replication Server time to empty its queues.	`300`	`60`

Parameter	Description	Example	Default
TIMER_INTERVAL	The number of seconds the RCM waits between server pings and monitoring commands. For example, if *MONITOR_WAIT* = 300 and *TIMER_INTERVAL* = 5, then the RCM issues the monitor command every 5 seconds for 5 minutes or until the switch active command completes at the Replication Server. The *TIMER_INTERVAL* must be less than both the *FAILOVER_WAIT* and *MONITOR_WAIT* parameters.	`10`	`5`
PING_TIMEOUT	The number of seconds the RCM attempts to verify that a server or database is available.	`4`	`3`
OSW_MONITOR_WAIT	The number of seconds that the RCM attempts to reconnect to an OpenSwitch server to which the RCM has lost its connection.	`15`	`5`
OSW_TIMER_INTERVAL	The number of seconds the RCM waits between attempts to reconnect to an OpenSwitch server to which the RCM has lost its connection.	`4`	`1`
DISCONNECT_STBY_USERS	If this parameter is set to 1 (true), users connected to the standby Adaptive Server are disconnected before application end users are switched to the standby Adaptive Server.	`1` (true)	`0` (false)
ASYNCHRONOUS	If this parameter is set to 1 (true), network communication is handled asynchronously. If the parameter is set to 0 (false), network communication is handled synchronously.	1 (true)	`0` (false)

Parameter	Description	Example	Default
RS_FAILOVER_MODE	This parameter determines the Replication Server failover strategy the RCM uses when the active Adaptive Server fails. Valid values are SWITCH, QUIESCE, or NONE. • SWITCH – the RCM issues the switch active command to the Replication Server. • QUIESCE– the RCM issues the suspend log transfer command and the admin quiesce_force_rsi command to quiesce the Replication Server. • NONE – the RCM does not issue any commands to the Replication Server, enabling you to manually perform fail over.	`SWITCH`	`SWITCH`
SWITCH_USERS	Determines whether or not the RCM switches the connections in the OpenSwitch server from active to standby after switching the Replication Server. If this parameter is set to 0 (false), the RCM does not switch the end users, enabling you to fail over manually. **Note** If not switched, the state of the active Adaptive Server in the OpenSwitch remains LOCKED.	`0` (false)	`1` (true)
NOTIFICATION_PROCESS	The name of a script or program that the RCM executes when an event occurs. See “Configuring the notification process” on page 144 for a list of events. This is an optional configuration parameter.	`email.sh`	None

Creating a redundant environment

To create a redundant high-availability, warm-standby environment, you must configure two OpenSwitch servers. One OpenSwitch server is the *primary*, which typically connects application end users to the active Adaptive Server. The second OpenSwitch server is the *secondary*, which typically connects DSS users to the standby Adaptive Server to load-balance the servers. In this case, the secondary OpenSwitch is never used by application end users unless the primary OpenSwitch fails.

See "DSS users" on page 119 for more information about a typical load-balancing environment.

To operate two OpenSwitch servers in your environment, you must also configure two RCM instances: The first RCM instance is the *primary* RCM, coordinating the connections for application end users; the second RCM instance is *redundant*, and is never used for failover processing unless the primary RCM fails.

Anticipating failures within a redundant environment

There are three important potential failures in a redundant environment:

- Failure of the primary OpenSwitch
- Failure of the secondary OpenSwitch
- Failure of the primary or redundant RCM

Failure of the primary OpenSwitch

The failure of the primary OpenSwitch, which means the loss of the connection between the two RCM instances and the primary OpenSwitch server, causes the following changes to the environment:

- After trying to reestablish the connection and failing, the primary RCM instance ceases execution.
- After trying to reestablish the connection to the primary OpenSwitch and failing, the redundant RCM instance assumes control of the failover operation.
- Users who connect to the environment through the primary OpenSwitch server (both application end users and DSS users) lose their connection to the primary OpenSwitch server and must log in again.

When these users log in again, they are connected to the secondary OpenSwitch server because it is the next entry in the *sql.ini* (Windows) or *interfaces* (UNIX) file record that describes the primary OpenSwitch server to these users. This multiple query entry in the *sql.ini* (Windows) or *interfaces* (UNIX) file enables user login connections to seamlessly roll over, or to change from the primary to the secondary OpenSwitch server.

See “Setting up the sql.ini or interfaces file” on page 132 for more information.

Note The RCM administrative login does not roll over during an OpenSwitch server failure. See “Setting up the sql.ini or interfaces file” on page 132 for more information about connection rollover.

Failure of the secondary OpenSwitch

The failure of the secondary OpenSwitch, which means the loss of the connection between the two RCM instances and the secondary OpenSwitch server, causes the following changes to the environment:

- After trying to reestablish the connection and failing, the primary RCM instance notes the failure of the secondary OpenSwitch server in its log.
- After trying to reestablish the connection to the secondary OpenSwitch and failing, the redundant RCM instance ceases execution.
- Because DSS users connect to the environment through the secondary OpenSwitch server, they lose their connection to the secondary OpenSwitch server and must log in again.

  When these users log in again, they are typically connected through the primary OpenSwitch server because it is the next entry in the *sql.ini* (Windows) or *interfaces* (UNIX) file record that describes the secondary OpenSwitch server to these users. The multiple query entry in the *sql.ini* (Windows) or *interfaces* (UNIX) file enables user logins to seamlessly roll over to the primary OpenSwitch.

See “Setting up the sql.ini or interfaces file” on page 132 for more information.

Note The RCM administrative login does not roll over during an OpenSwitch server failure. See “Setting up the sql.ini or interfaces file” on page 132 for more information.

Failure of the primary and redundant RCM instances

Failure of an RCM instance is unlikely; however, you should be prepared for its potential failure because it can mean the loss of failover capability of the environment. The failure of the primary RCM can mean that your environment no longer has the capability of failing over in a catastrophic situation because the primary RCM is no longer running and no longer aware of the status of the system. Similarly, the failure of the redundant RCM can mean the loss of the RCM’s overall ability to detect the failure of the primary OpenSwitch server because the redundant RCM could not then assume control of failover if the primary OpenSwitch server fails.

To gain some protection from an RCM failure, you must set the *COORD_MODE* parameter to “ALWAYS” in the OpenSwitch configuration file. This ensures that any logins to an OpenSwitch server after an RCM failure. This login failure notifies users of a problem so that you can take steps to recover, such as stopping and restarting servers.

See “Setting up the sql.ini or interfaces file” on page 132 for more information about OpenSwitch configuration parameters.

Configuring two OpenSwitch servers

To use two OpenSwitch servers effectively in a warm-standby environment, you must use features provided by both OpenSwitch and the connectivity software. You can then configure your environment so that users are switched from a primary OpenSwitch server to the secondary OpenSwitch server upon failover. This is also described in this section as a rollover of the connections.

Note You can use the redundant RCM to funnel DSS users to the standby Adaptive Server.

Configuring a redundant environment is complex, and you must be aware of the following constraints:

- You must add an entry to the *sql.ini* (Windows) or *interfaces* (UNIX) file for each OpenSwitch server; one for the primary OpenSwitch server and one for the secondary OpenSwitch server.
- You must add a second query line to each OpenSwitch server entry that contains redundant connection information to be used during a rollover.
- The configuration files for the primary RCM and redundant RCM instances can be identical.

  See “Setting up a configuration file for two RCM instances” on page 136 to view an example of an RCM configuration file for a redundant OpenSwitch environment.
- If you use batch files to run the RCM, you must create two batch files or scripts, one for each RCM instance.
- The redundant RCM must be started at the command line using the rcm command with the *-R* flag. See “Starting and stopping the RCM” on page 146 for more information about the *-R* flag.

Setting up the *sql.ini* or *interfaces* file

Load balancing

To load-balance your system among application end users and DSS users, you must add an entry to the *sql.ini* (Windows) or *interfaces* (UNIX) file for each OpenSwitch server, giving each entry a unique name. The entry for the primary OpenSwitch server is used only by application end users; the entry for the secondary OpenSwitch server is used only by DSS users.

A redundant OpenSwitch

A redundant OpenSwitch environment requires two query lines for each OpenSwitch server entry in the *sql.ini* (Windows) or *interfaces* (UNIX) file. This feature enables the automatic rollover of users connecting to a server.

The order of the query lines describes the sequence in which the connectivity software attempts to connect to a specific server when a user logs in. If the connection attempt using the first query line in that server’s entry fails, the software tries to connect using the next query line. This rollover of user connections gives you redundant connectivity through OpenSwitch.

See the *Open Client DB-Library/C Reference Manual* for more information about placing multiple query entries in your *sql.ini* (Windows) or *interfaces* (UNIX) file.

Redundant RCMs

A completely redundant environment features two RCM instances: One coordinates failover; the other waits to take over coordination if the first instance fails.

Warning! If you use multiple query lines for the RCM connections to the primary and secondary OpenSwitch servers, your designed failover might be disrupted when you start the RCM. This situation could be catastrophic.

If you use multiple query lines for the RCM connections, and the primary RCM instance successfully connects to the secondary OpenSwitch server after attempting to connect to a failed or nonexistent primary OpenSwitch server, the primary RCM instance is no longer coordinating failover through the primary OpenSwitch server, because it is connected to the secondary OpenSwitch server. In this scenario, neither the RCM nor the system administrator is aware of the change in OpenSwitch servers, but now the RCM is connected to an OpenSwitch server that is configured to be the secondary OpenSwitch server, not the primary OpenSwitch server.

To resolve this issue, you must create two *sql.ini* (Windows) or *interfaces* (UNIX) file entries for each OpenSwitch server and give each entry a unique server name. The first *sql.ini* (Windows) or *interfaces* (UNIX) file entry is for end users and should include a second query line for automatic rollover to the secondary OpenSwitch in case the primary OpenSwitch fails, as follows:

```
[usr_os1]
master=TCP,tokyo,2000
query=TCP,tokyo,2000
query=TCP,newyork,2900

[usr_os2]
master=TCP,newyork,2900
query=TCP,newyork,2900
query=TCP,tokyo,2000
```

In this *sql.ini* (Windows) or *interfaces* (UNIX) file example, the primary OpenSwitch server, “os1,” runs on port number 2000 on the computer “tokyo.” The secondary OpenSwitch server, “os2,” runs on port number 2900 on “newyork.” The *sql.ini* (Windows) or *interfaces* (UNIX) file records, “usr_os1” and “usr_os2,” designate the primary and secondary OpenSwitch servers, respectively, used by application end users and DSS users. Because of the dual query lines in the entry, a user logging in to a failed OpenSwitch server is automatically connected, or rolled over, to the secondary OpenSwitch server.

The second *sql.ini* (Windows) or *interfaces* (UNIX) file record is for the two RCM administrator logins and should include only one query entry for the primary OpenSwitch server to ensure that the primary RCM instance connects only to the primary OpenSwitch server. It also includes only one query entry for the secondary OpenSwitch server to ensure that the redundant RCM instance connects only to the secondary OpenSwitch server. This is an example of the second *sql.ini* (Windows) or *interfaces* (UNIX) file record:

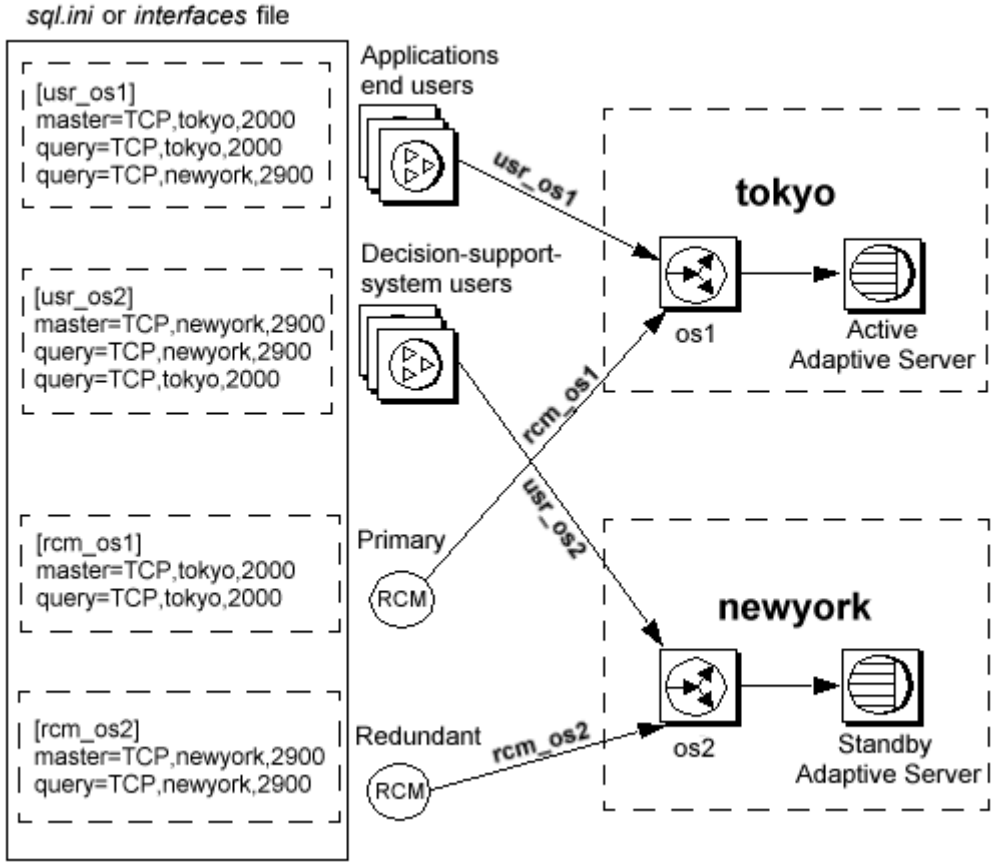
```
[rcm_os1]
master=TCP,tokyo, 2000
query=TCP,tokyo, 2000

[rcm_os2]
master=TCP,newyork,2900
query=TCP,newyork,2900
```

The RCM administrator logins use the server names “rcm_os1” and “rcm_os2” to connect to the two OpenSwitch servers (also identified in the RCM configuration file). Because each OpenSwitch server record that the RCM administrator logins use contains only one query entry, each RCM administrator login connection does not roll over to another query entry like a user login connection would.

For example, if the primary OpenSwitch server, rcm_os1, is not running, the primary RCM instance cannot run. The primary RCM instance does not know to connect to the secondary OpenSwitch, rcm_os2, because it is not indicated in the server record for the primary OpenSwitch server. This enables you to identify a problem with the primary OpenSwitch server or with the connection between the RCM and the OpenSwitch server rather than have the RCM administrative login roll over automatically to the second OpenSwitch server without notifying you. It also ensures that an RCM instance is either in control of failover or it ceases to run. Because end users are now connected through the secondary OpenSwitch server to the back-end database, you can respond manually to the failure when it is convenient to the end users.

Figure 4-3: sql.ini or interfaces file for a redundant environment



Note To create a redundant environment, create entries for both the primary and redundant RCM instances in your *sql.ini* (Windows) or *interfaces* (UNIX) file, as well as entries for both the primary and secondary OpenSwitch servers.

See "RCM configuration file examples" on page 137 to find an example of an RCM configuration file used in an environment with two OpenSwitch servers.

Setting up two OpenSwitch configuration files

In the configuration files for both the primary and secondary OpenSwitch servers, you must set the status to UP. To do this, use a text editor to open the configuration file, locate the [SERVER] section, and next to the *STATUS* parameter, enter "UP". If the secondary OpenSwitch server is up, it can allow DSS users to access the environment, enabling load-balancing or login control.

See "DSS users" on page 119 for more information.

Setting up a configuration file for two RCM instances

The two RCM instances in a redundant environment can use the same configuration file. To set up this file properly, add the name of the secondary OpenSwitch server as well as the name of the primary OpenSwitch server to the "OpenSwitch Server" section of the RCM configuration file, as shown:

```
OPENSWITCH = rcm_os1
SECONDARY_OPENSWITCH = rcm_os2
```

Where the primary OpenSwitch server is "rcm_os1," and the secondary OpenSwitch server is "rcm_os2."

See "Understanding RCM configuration parameters" on page 122 for more information about RCM configuration parameters.

Command line flag for the redundant RCM

To distinguish the redundant from the primary RCM instance, use the -R flag with the rcm command at the command line when you start the redundant RCM. When you use the -R option, the redundant RCM:

- Does not perform a failover upon detection of database or Replication Server problems
- Handles the command processing for the secondary OpenSwitch
- Assumes the control of the failover process upon the loss of its connection to the primary OpenSwitch
- Restricts application end users' access unless the primary OpenSwitch server fails

Starting OpenSwitch and the RCM after OpenSwitch failure

To restart an OpenSwitch server and an RCM instance after the failure of an OpenSwitch server in a redundant environment and, therefore, after the failure of the corresponding RCM instance:

1 Restart the failed OpenSwitch server.

2 Stop the RCM instance that is still running (the redundant RCM instance).

3 Restart the primary and redundant RCM instances (using the -R command line option for the redundant RCM).

This ensures that:

- Only one RCM instance is controlling failover
- All RCM connections are reestablished to both OpenSwitch servers
- User logins to both OpenSwitch servers are handled by the appropriate RCM instance

RCM configuration file examples

This section includes examples of RCM configuration files designed for different purposes within a high-availability, warm-standby environment.

Each configuration parameter is described in “Understanding RCM configuration parameters” on page 122.

See “Setting up a configuration file for two RCM instances” on page 136 to find an example of an RCM configuration file for a redundant environment.

Failover modes

This section shows examples of the RCM configuration file for each failover mode you can choose: switch active, quiesce, and none.

switch active

This section shows an example of an RCM configuration file for switch active mode:

```
# Open Switch Server
OPENSWITCH = ws_os
COORD_USER = os_coord
COORD_PASSWORD = os_coord_pwd
```

```
# Replication Server
REP_SERVER = ws_rs
RS_USER = sa
#RS_PASSWORD - Replication Server password is blank

# Active and Standby ASE Servers
ACTIVE_ASE = BookServer
ACTIVE_USER = sa
#ACTIVE_PASSWORD - ACTIVE ASE password is blank

STANDBY_ASE = StandbyBook
STANDBY_USER = stndby_sa
STANDBY_PASSWORD = booknut

# On failover switch the flow of replication
RS_FAILOVER_MODE = SWITCH

# Identify the databases in the warm-standby environment
LOGICAL_CONN = LDS.LDB
ACTIVE_DBS = pubs3
STANDBY_DBS = pubs3
REQUIRED_DBS = pubs3

APP_POOL= Application

# Wait 5 minutes before starting the failover
FAILOVER_WAIT = 300

# Provide Replication Server 2 minutes perform the
switch active
MONITOR_WAIT = 120
```

If you use this example configuration file in your environment and the active Adaptive Server fails, the RCM switches the logical connection named "LDS.LDB" in the Replication Server. Then the RCM starts the Replication Agent thread in the standby Adaptive Server for the database "pubs3."

quiesce

This section shows an example of an RCM configuration file for quiesce mode.

The section highlighted in bold is the only difference between this example and the previous example configuration file for switch active mode. See "switch active" on page 137 for comparison.

```
# Open Switch Server
```

```
OPENSWITCH = ws_os
COORD_USER = os_coord
COORD_PASSWORD = os_coord_pwd

# Replication Server
REP_SERVER = ws_rs
RS_USER = sa
#RS_PASSWORD - Replication Server password is blank

# Active and Standby ASE Servers
ACTIVE_ASE = BookServer
ACTIVE_USER = sa
#ACTIVE_PASSWORD - ACTIVE ASE password is blank

STANDBY_ASE = StandbyBook
STANDBY_USER = stndby_sa
STANDBY_PASSWORD = booknut

# On failover quiesce the Replication Server
# No database information is needed
RS_FAILOVER_MODE = QUIESCE

# Test to make sure that the pubs3 database is available
REQUIRED_DBS = pubs3

APP_POOL= Application

# Wait 5 minutes before starting the failover
FAILOVER_WAIT = 300

# Provide Replication Server 2 minutes perform the
switch active
MONITOR_WAIT = 120
```

If you use this example configuration file in your environment and if the active Adaptive Server fails, the RCM issues the quiesce command to Replication Server. All connections in Replication Server are then quiesced.

none

This section shows an example of an RCM configuration file for none mode.

The section highlighted in bold is the only difference between this example and the previous example for SWITCH ACTIVE mode. See “switch active” on page 137 for comparison.

```
# Open Switch Server
```

```
OPENSWITCH = ws_os
COORD_USER = os_coord
COORD_PASSWORD = os_coord_pwd

# No Replication Server information is needed

# Active and Standby ASE Servers
ACTIVE_ASE = BookServer
ACTIVE_USER = sa
#ACTIVE_PASSWORD - ACTIVE ASE password is blank

STANDBY_ASE = StandbyBook
STANDBY_USER = stndby_sa
STANDBY_PASSWORD = booknut

# Manual Replication Server failover
# No database information is needed
RS_FAILOVER_MODE = NONE

# Don't switch the users to the standby ASE
SWITCH_USERS = 0

APP_POOL = Application

# Wait 5 minutes before starting the failover
FAILOVER_WAIT = 300

# Provide Replication Server 2 minutes perform the
switch active
MONITOR_WAIT = 120
```

If you use this example configuration file in your environment and the active Adaptive Server fails, the RCM takes no action for Replication Server.

Multiple databases

This section shows an example of an RCM configuration file set up to support multiple databases in a warm-standby environment. In a multiple database environment, an Adaptive Server contains more than one database involved in warm-standby replication.

An Adaptive Server can contain several databases that are each being replicated to the standby Adaptive Server. When the active Adaptive Server fails, each database connection must be switched to the standby Adaptive Server.

Following are the results for each failover mode:

- switch active – each database connection in the *LOGICAL_CONNECTION* parameter is switched to the standby Adaptive Server.
- quiesce – by default all database queues in Replication Server are emptied before the database connections are switched to the standby Adaptive Server.
- none – multiple databases are treated the same way as a single database—you are notified of the failure.

The section highlighted in bold is the only difference between this example and the previous example for switch active mode with a single logical database. See "switch active" on page 137 for comparison.

```
# Open Switch Server
OPENSWITCH = ws_os
COORD_USER = os_coord
COORD_PASSWORD = os_coord_pwd

# Replication Server
REP_SERVER = ws_rs
RS_USER = sa
#RS_PASSWORD - Replication Server password is blank

# Active and Standby ASE Servers
ACTIVE_ASE = BookServer
ACTIVE_USER = sa
#ACTIVE_PASSWORD - ACTIVE ASE password is blank

STANDBY_ASE = StandbyBook
STANDBY_USER = stndby_sa
STANDBY_PASSWORD = booknut

# On failover switch the flow of replication
RS_FAILOVER_MODE = SWITCH

# Identify the databases in the warm-standby environment
LOGICAL_CONN = LDS.pubs3, LDS.sales, LDS.signings

#DATABASES - Omitted, so RCM will use pubs3, sales,
signings

# The loss of the signings database will not trigger a
failover
REQUIRED_DBS = pubs3, sales
```

```
APP_POOL = Application

# Wait 5 minutes before starting the failover
FAILOVER_WAIT = 300

# Provide Replication Server 2 minutes perform the
switch active
MONITOR_WAIT = 120
```

If you use this example configuration file in your environment and the active Adaptive Server fails, the RCM takes the same action as in the switch active mode example (see "switch active" on page 137), but switches all logical connections listed in the *LOGICAL_CONN* parameter. That is, the RCM switches the logical connections named "LDS.pubs3," "LDS.sales," and "LDS.signings" in the Replication Server (one connection for each database). Then the RCM starts a Replication Agent in the standby Adaptive Server for each of the pubs3, sales, and signings databases. The *DATABASES* parameter is omitted, so that the RCM uses the database names identified in the *LOGICAL_CONN* parameter when starting the Replication Agents on the standby Adaptive Server. In this example, the *REQUIRED_DBS* parameter does not include the signings database; therefore, a failure in that database does not trigger the failover process.

Tuning

If you have set the *RS_FAILOVER_MODE* parameter to QUIESCE or SWITCH, RCM monitors the Replication Server during a failover process. The RCM monitors the failover process to determine when the Replication Server commands switch active or suspend log transfer have completed.

Certain configuration parameters control how RCM monitors the failover process:

- *FAILOVER_WAIT* – the number of seconds the RCM waits after a potential failover is detected before initiating the failover process. This failover waiting period gives the active Adaptive Server an opportunity to recover automatically. For example, if you set *FAILOVER_WAIT* to 60, RCM waits 60 seconds before initiating the failover process.

- *MONITOR_WAIT* – the number of seconds the RCM monitors Replication Server after invoking a failover command in Replication Server and before switching end users to the standby Adaptive Server. This gives the Replication Server time to empty its queues. For example, if you set *MONITOR_WAIT* to 60, RCM monitors Replication Server for 60 seconds.

  Note The *MONITOR_WAIT* parameter is not used if the *RS_FAILOVER_MODE* parameter is set to NONE.

- *TIMER_INTERVAL* – the number of seconds the RCM waits between server pings and monitoring commands. The *TIMER_INTERVAL* value must be less than or equal to the values of the *FAILOVER_WAIT* and *MONITOR_WAIT* parameters.

  For example, if you set *TIMER_INTERVAL* to 5, RCM waits 5 seconds between server pings and monitoring commands. If you set *FAILOVER_WAIT* to 60, the RCM pings the server 12 times before beginning the failover process.

  Note If the *TIMER_INTERVAL* value is greater than either or both the *FAILOVER_WAIT* and *MONITOR_WAIT* values, the RCM does not start and displays a notification that there is an error in the parameter settings.

You can tune the system using these configuration parameters. Used together, these parameters work as described in the following scenario:

1 The RCM detects a failover in the system.

2 RCM pings the active Adaptive Server every *TIMER_INTERVAL* seconds for *FAILOVER_WAIT* seconds to determine if it has recovered.

3 After *FAILOVER_WAIT* seconds, the Adaptive Server has not recovered, so the RCM initiates the failover process. The RCM begins to monitor Replication Server. Every *TIMER_INTERVAL* seconds, the RCM issues a monitoring command.

4 The RCM continues to monitor Replication Server for *MONITOR_WAIT* seconds. At that time, or when the Replication Server finishes the failover process if that is sooner, the RCM switches the users to the standby Adaptive Server.

Note The RCM uses the *FAILOVER_WAIT* and *TIMER_INTERVAL* parameters to monitor the environment even when you set the *RS_FAILOVER_MODE* parameter to "NONE" because you plan to fail over the Replication Server manually. In this case, the RCM responds to a failover by locking user connections out of the Adaptive Server, but does not invoke any Replication Server commands.

Configuring the notification process

The RCM can execute a process when an event occurs, for example, when failover begins. This process is a script or a program that you create and then define in the RCM configuration file using the *NOTIFICATION_PROCESS* parameter.

When working with the RCM notification process, be aware that:

- The notification process is executed from the RCM current working directory (the directory where the RCM executable is installed).
- The notification process is executed with the same set of permissions used to execute the RCM.
- Output is redirected to a temporary file. The full path name of this file is written to the RCM log and is prefixed with "rcm."
- The RCM does not delete the temporary output file.
- The notification ID and text message are passed as parameters to the script program.

Table 4-5 lists the events that trigger the RCM notification process.

Table 4-5: Notification process events

Notification ID	Event description
1	The RCM has detected a possible failover situation where the active Adaptive Server is not responding. The RCM is about to enter a wait state to determine if the active Adaptive Server is down, or if it will automatically recover.

Notification ID	Event description
2	The failover process has started. The RCM has determined that the active Adaptive Server is down and is failing over in the replication environment.
3	The failover has been aborted, because the active Adaptive Server has recovered before the RCM started the failover process.
4	The RCM cannot connect to Replication Server. The RCM switches the users to the standby Adaptive Server without failing over the logical connections in Replication Server.
5	The RCM cannot start the Replication Agent in the standby Adaptive Server after switching the logical connections in Replication Server.
6	Executing the failover process in Replication Server has failed. This occurs when the RCM unsuccessfully executes either the switch active or the quiesce commands.
7	The RCM has finished the Replication Server failover process. The RCM is about to switch users to the standby Adaptive Server.
8	Switching the users in OpenSwitch from the active Adaptive Server to the standby Adaptive Server has failed. The active Adaptive Server is locked, all existing connections are suspended, and new users cannot log in.
9	The RCM failover process has completed.
10	The RCM has exited, probably because the connection to the primary OpenSwitch is lost, or because of some internal error.
11	The RCM has lost the connection to the OpenSwitch. The message identifies which OpenSwitch server failed. The RCM exits if the OpenSwitch server is not a secondary OpenSwitch server in a dual OpenSwitch environment. **Note** In a dual OpenSwitch environment, if the primary OpenSwitch fails, the RCM sends this notification and notification ID 10 to the administrator, then exits. If the secondary OpenSwitch fails, the RCM sends this notification ID 11 to the administrator, but does not exit.
12	A test notification is executed when the user starts the RCM with the -a (analyze) option.

The RCM displays a notification ID and a text message in its log when any of the events in Table 4-5 on page 144 occur. The text message is similar to the event description in the table. To determine what process you want performed when events occur, read the event description in Table 4-5 on page 144.

Some examples of processes that can be triggered by events are:

- Launching e-mail
- Launching a pager program
- Running a script that displays the notification ID and event description to console

For example, the following segment of the configuration file sets the *NOTIFICATION_PROCESS* parameter to execute a program called *email.sh* when an event occurs:

```
# Set notification process to email me
NOTIFICATION_PROCESS  = email.sh
```

To set up the notification process to trigger a program or script, you must set the *NOTIFICATION_PROCESS* parameter. If you do not include this parameter, the RCM does not send notification of events. See "Understanding RCM configuration parameters" on page 122 for more information.

The RCM records all events in its log, so if you do not set the *NOTIFICATION_PROCESS* parameter but need to troubleshoot the failover, examine the RCM log, called *rcm.log*, in the RCM subdirectory.

Starting and stopping the RCM

You can start the RCM:

- Automatically from OpenSwtich after OpenSwitch starts. See "Starting and stopping the RCM automatically from OpenSwitch" on page 146.
- From the command line. See "Starting an RCM at the command line" on page 147.
- Using a batch or script file. See your operating system documentation for more information about creating batch or script files.

Note If you start the RCM from the command line or from a script file, the OpenSwitch server must be running before you can start the RCM. See the *OpenSwitch Administration Guide* for more information about starting OpenSwitch.

Starting and stopping the RCM automatically from OpenSwitch

OpenSwitch version 15.0 and later allows you to start and stop the RCM automatically from OpenSwitch when OpenSwitch starts.

To configure this functionality, see the *OpenSwitch Administration Guide*, Chapter 3, "Starting and Stopping OpenSwitch and RCMs," for instructions.

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. See the *OpenSwitch Administration Guide*, Chapter 3, "Starting and Stopping OpenSwitch and RCMs.
- Registered procedures rp_rcm_startup, rp_rcm_shutdown, rp_rcm_connect_primary, and rp_rcm_list. See the *OpenSwitch Administration Guide*, Chapter 6, "Registered Procedures," for details.
- *RCMNAME* parameter in the RCM configuration file. See Table 4-4 on page 123.

Starting an RCM at the command line

You must start OpenSwitch before starting the RCM.

To start an RCM at the command prompt, enter:

```
rcm -c config_file -e system_log -i sql.ini_or_interfaces_file
```

Note You cannot start the RCM as a Windows service.

Syntax

This section describes the command syntax and command line flags you can set at RCM start-up.

rcm [-v] [-h] [-a] [-R]
[-c *config_file*]
[-e *system_log*]
[-i *sql.ini or interfaces_file*]
[-T *trace_flags*]
[-E *filename*]

Command line flags

- -v – prints the version number and the copyright message, then exits.
- -h – prints the help message and exits.
- -a – analyzes the replication environment and exits. The RCM:
  - Tests the configuration parameters and prints the results to *stdout*
  - Validates all configuration parameters

- Connects to the OpenSwitch server
- Logs in to the active Adaptive Server
- Verifies that the active databases exist
- Logs in to the standby Adaptive Server
- Verifies that the standby databases exist
- Logs in to the Replication Server
- Verifies that the logical connection exists
- Tests the ranges of the tuning parameter values

Note See "Tuning" on page 142 for more information about setting these values.

- Tests the notification process if the *NOTIFICATION_PROCESS* parameter is set (see "Understanding RCM configuration parameters" on page 122)
- Prints out all configuration parameters

- -R – indicates that the current instance of RCM is redundant. When you use the -R flag, you indicate that the redundant RCM does not perform a failover, handles command processing for the secondary OpenSwitch, and assumes the control of failover if it loses its connection to the primary OpenSwitch.
- -c *config_file* – the full path name of the RCM configuration file. If you omit the -c flag, the RCM looks for a configuration file named *rcm.cfg* in the current directory.
- -e *system_log* – the full path name of the system log file. The RCM writes all system, error, and trace messages to the system log file. If you omit the -e flag, the RCM writes messages to a file named *rcm.log* in the current directory.
- -i *sql.ini_or_interfaces_file* – the full path name of the Sybase *sql.ini* (Windows) or *interfaces* (UNIX) file that the RCM searches when connecting to servers. If you omit the -i flag, the RCM looks for the *sql.ini* (Windows) or *interfaces* (UNIX) file in the directory to which the SYBASE environment variable points.

  On UNIX, the default *interfaces* file is in the Sybase installation directory (*$SYBASE*). On Windows, the default *sql.ini* is in *%SYBASE%/ini*.

- -T *trace_flags* – this flag sets trace flags in the RCM. Use this flag to debug your environment. Following is the list of valid trace flags. To set more than one flag, use a comma-separated list; for example, `-T A,C,F`.

A	Set all trace flags.
C	Displays information on connectivity issues, including connecting and disconnecting from servers and executing commands.
E	Traces execution of the RCM when resolving OpenSwitch connection issues. It also displays end-user connection information, such as user, application, requested server, and so on.
F	Traces the execution of the failover process when the RCM coordinates the switching from the active and standby databases.
G	Displays general or miscellaneous information.
I	Traces the RCM's initialization steps, including reading the configuration file, installing the callback handlers, and connecting to the OpenSwitch.
M	Writes all messages generated by the RCM to the system log. These messages include all connectivity messages and all messages sent to the RCM by Replication Server and Adaptive Server.
N	Displays all notification process messages.
O	Writes OpenSwitch server messages and connectivity messages generated from connections between the RCM and the OpenSwitch to the system log. Situations generating these messages are usually handled by the RCM, so these messages are typically redundant.
R	Traces the process of coordinating multiple OpenSwitch servers.
S	Writes all commands that the RCM sends to other servers to the system log.

- -E *filename* – user name and password encryption. You can provide an optional *filename* argument. If you provide a filename, that file is created and the encrypted user names and passwords are written to that file and to the console. If you do not provide a filename, the encrypted user names and passwords are written only to the console.

Stopping the RCM manually

To shut down an RCM from the command line, use rp_rcm_shutdown. See cm_rp_rcm_shutdown on page 93 for details.

Note If the RCM detects an error, it shuts down automatically, posting a notification message to the log. See “Configuring the notification process” on page 144.

Recovering from a coordinated failover

When a failover occurs in your environment, you must recover the active-standby setup.

Note When you use the NONE option to create manual failover, you must develop your own recovery procedures.

Recovering from switch active failover

When you use the SWITCH mode to fail over automatically, you must restart the active server and resume database connections, and so on, to recover the high-availability environment. This section describes the steps you must take to do this.

Table 4-6 on page 151 uses the following acronyms:

- ADB – the active database name
- ADS – the active data server name
- SDB – the standby database name
- SDS – the standby data server name
- LDS – the logical data server name
- LDB – the logical database name

Table 4-6: Steps to recover from automatic failover

Step	Server	Example command line
1) Start the active Adaptive Server.	Active Adaptive Server	On Windows, `run [Active_ASE].bat` On UNIX, `dataserver -d [master_db_file] -R -c [config_file].`
2) Resume the active Replication Server connections.	Replication Server	`resume connection to <ADS>.<ADB>`
3) Suspend connections to the standby Adaptive Server.	OpenSwitch	`rp_server_status <SDS>, 'LOCKED' rp_stop NULL, <SDS>, NULL, 1, 1`
4) Monitor the Replication Agent thread.	Standby Adaptive Server	`sp_help_rep_agent <SDB>, 'scan'` or watch for the last transaction in the active Adaptive Server
5) Stop the Replication Agent thread.	Standby Adaptive Server	`sp_stop_rep_agent <SDB>`
6) Switch the Replication Server connections.	Replication Server	`switch active connection for <LDS>.<LDB> to <ADS>.<ADB>`
7) Monitor the switching process.	Replication Server	`admin logical_status, <LDS>.<LDB>`
8) Start Active Adaptive Server Replication Agent.	Active Adaptive Server	`sp_start_rep_agent <ADB>`
9) Switch users back to active Adaptive Server.	OpenSwitch	`rp_server_status <ADS>, 'UP' rp_server_status <SDS>, 'UP' rp_switch NULL, <ADS>, NULL-1, <ADS> rp_start NULL, NULL, NULL`
10) Resume the standby Replication Server connections.	Replication Server	`Resume connection to <SDS>.<SDB>`

Unexpected failure of Replication Server

If you are unaware that Replication Server has stopped running, the database environment may become corrupted. If the Adaptive Server fails and the RCM attempts to connect to Replication Server, which also fails, the standby server is out of date because there was a period of time during which the Replication Server was down and not replicating transactions to the standby server.

Attempts to add transactions to the standby server at this point might fail, and, as a result, the entire database environment could be out of date. In this case, the RCM still switches users to the standby environment to ensure that current transactions are being captured. If the entire database environment becomes out of date, you must recover from backup, following your internal procedure for recovery.

See "Managing a Warm Standby Environment" in *Replication Server Administration Guide, Volume 2* for more information about transaction processing in a warm-standby environment.

Troubleshooting

This section describes some procedures you can use to help troubleshoot problems with the high-availability, warm-standby environment.

Analyzing the RCM environment

You can use the "-a" flag with the rcm command to analyze your environment. To analyze the RCM environment, enter rcm -a at the command line.

This is example output from the rcm -a command.

```
Writing to the system log file: 'rcm.log'.
Reading the configuration file: 'rcm.cfg'.
Using the 'us_english' language.
Using the 'iso_1' character set.
OpenSwitch server name: 'ws_os'.
OpenSwitch coordination module username: 'os_coord'.
Active ASE server name: 'BookServer'.
Active ASE username: 'sa'.
Standby ASE server name: 'StandbyBook'.
Standby ASE username: 'stndby_sa'.
OpenSwitch application pool name: 'Application'.
RCM is configured to wait 300 seconds before initiating the failover process.
RCM is configured to monitor the Replication Server failover for 120 seconds.
The RCM timer interval is configured to be 5 seconds.
The RCM will not disconnect users from the standby ASE on a failover.
The RCM will switch users to the standby ASE after a failover.
The RCM will issue the host ping command before attempting to connect to a
server.
```

```
Ping Host Command: 'ping'.
Replication Server failover mode: 'SWITCH'.
Replication Server host name: 'StndbySun8'.
Replication Server name: 'ws_rs'.
Replication Server username: 'sa'.
Logical connection list: LDS.LDB
Active Database list: pubs3
Standby Database list: pubs3
Required database list: pubs3
Attempting to initialize the coordination module's connectivity.
The coordination module's connectivity initialized successfully.
Attempting to create the coordination module.
The coordination module was created successfully.
Attempting to connect to the OpenSwitch 'ws_os', username 'os_coord'.
Connected to the OpenSwitch 'ws_os', username: 'os_coord'.
Attempting to connect to the ASE Server 'BookServer', username 'sa'.
Successfully connected to the ASE server 'BookServer'.
Logged into the database 'pubs3'.
Attempting to connect to the ASE Server 'StandbyBook', username 'stndby_sa'.
Successfully connected to the ASE server 'StandbyBook'.
Logged into the database 'pubs3'.
A standby OpenSwitch was not defined.
Attempting to initialize connectivity for the Replication Server.
The connectivity to the Replication Server was initialized successfully.
Attempting to connect to the Replication Server 'ws_rs', username 'sa'.
Connected to the Replication Server 'ws_rs'.
Attempting to retrieve the status of the logical connections.
Logical connection 'LDS.LDB'.
Active connection 'BookServer.pubs3', State: 'Suspended/'.
Standby connection 'StandbyBook.pubs3', State: 'Active/'.
Current operation: 'None', Step: 'None'.
```

See “Starting an RCM at the command line” on page 147 for details about the “-a” flag.

Monitoring the environment with Replication Server plug-in

You can use the Replication Server plug-in, which comes with Replication Server, to monitor the environment, including viewing the Replication Server log.

See Appendix B, “Getting Started with RSM,” in the *Replication Server Installation Guide* for your platform for more information about using the Replication Server plug-in.

RCM internal coordination

This section describes how the RCM coordinates failover in a high-availability, warm-standby environment.

The RCM start-up process

When the RCM starts, it:

1 Reads the command line parameters.

2 Reads and validates the RCM configuration file parameters.

3 Logs start-up information: version string, copyright, and critical configuration parameters.

4 Connects to OpenSwitch.

5 Monitors user connections to OpenSwitch.

OpenSwitch connection coordination

The responsibility of a CM is to coordinate the end-user connections that pass through the OpenSwitch to the Adaptive Servers. OpenSwitch notifies the coordination module whenever:

- A user attempts to connect to OpenSwitch.
- An attempt fails.
- An existing connection to an Adaptive Server fails

.

In this way, the RCM coordinates the switch of users to a different server through OpenSwitch.

End-user login request

When an end user requests a connection, the connectivity software establishes a connection to the first available OpenSwitch server. When an OpenSwitch server fails, end-user connections are dropped. When the end user reconnects, the connectivity software establishes a connection to the alternate OpenSwitch server.

OpenSwitch CMs process end-user login requests to servers controlled by OpenSwitch. When the RCM receives a login request, it tells the OpenSwitch server to log the user in to the requested server. It does not determine if the server is available or if a failover process has occurred. If OpenSwitch determines that the server is not available, it sends the RCM a login failure notification (see "End-user login or connection failure" on page 155). After the RCM has processed the failure, OpenSwitch changes the server status to DOWN and requests a connection to the standby server.

Note When processing a login request, the RCM does not distinguish between an application end user and a DSS user. Only upon login request failure does the RCM note the type of user requesting to log in. If an application end-user login fails, the RCM begins the failover process. See "Application end users" on page 119.

If the Open Switch server fails, your environment is protected because user logins are switched to the secondary OpenSwitch server.

End-user login or connection failure

The RCM is notified of an Adaptive Server failure when login requests to the Adaptive Servers fail, or when existing connections to the Adaptive Servers fail. Depending on the type of end user and the Adaptive Server, the RCM performs the following processes:

- Active Adaptive Server – if an application end-user connection fails, the OpenSwitch server notifies the RCM. If the active Adaptive Server has failed, the RCM starts the failover process. All application end-user connections are suspended until the failover process is finished.

  If a DSS user connection fails, the OpenSwitch notifies the RCM. If the active Adaptive Server fails, the RCM routes the connection to the next available server. If there is no "next" server because the other server in the environment is down, the RCM logs an error message. Because DSS users are read-only, the RCM switches them to the standby server without starting the failover process.

- Standby Adaptive Server – if an application end-user connection fails, OpenSwitch notifies the RCM. If the standby Adaptive Server fails, the RCM routes the connection to the next available server. If there is no "next" server because the other server in the environment is down, the RCM logs an error message. In this scenario, application end users are working on the standby server because the active server has already failed. The RCM cannot continue to route users unless the active Adaptive Server is running again and able to take login requests.

  Note The RCM and Replication Server support fail over to two servers only.

  If a DSS-user connection fails, the OpenSwitch notifies the RCM. If the standby Adaptive Server fails, the RCM routes the connection to the next available server. This can include routing the DSS user to the active Adaptive Server. If there is no "next" server because the other server in the environment is down, the RCM logs an error message.

See "Failover processing" on page 156 for more details about failover.

See "End-user connectivity" on page 112 for more information about users and user connections.

Failover processing

When notified of a failed connection, the RCM performs the following tasks:

1 Before starting the failover process, RCM pings the active Adaptive Server. If the RCM can ping the Adaptive Server server, it is not down, so the RCM issues a kill command to end the current connection. The end user must manually reconnect.

2 The RCM changes the status of the active Adaptive Server in the OpenSwitch log to LOCKED. This stops new users from connecting to the active Adaptive Server.

3 The RCM issues a stop command to suspend all current connections to the active Adaptive Server.

4 The RCM does not fail over immediately but waits to see if the system recovers. The Adaptive Server might automatically recover, or the network might stabilize. The RCM pings the active Adaptive Server at a configurable interval. If the RCM successfully pings the server, it unlocks the server, restarts the connections, and allows users to connect.

5 When RCM determines that a failover is necessary, it performs the following steps:

- If *RS_FAILOVER_MODE* is set to SWITCH, the RCM connects to the Replication Server and issues the switch active command for each logical connection defined by the *LOGICAL_CONN* configuration parameter.
- If *RS_FAILOVER_MODE* is set to QUIESCE, the RCM connects to Replication Server and issues the suspend log transfer from all and admin quiesce_force_rsi commands.
- If the *RS_FAILOVER_MODE* is set to NONE, the RCM does not connect to Replication Server, but locks out user connections to the Adaptive Server.

6 When *RS_FAILOVER_MODE* is not set to NONE, because both the switch active command and the quiesce commands are asynchronous, the RCM monitors the process to determine when the commands have completed. The RCM issues a monitoring command at a configurable interval until a configurable amount of time is reached. At that time, or when Replication Server finishes the failover process, whichever occurs first, the RCM switches the users to the standby Adaptive Server.

Note The monitoring commands the RCM issues are different for switch active and quiesce modes. In switch active mode, the RCM issues the admin logical status command. In quiesce mode, the RCM issues the admin health command.

---

7 If *RS_FAILOVER_MODE* is set to SWITCH, the RCM starts the Replication Agent on the standby Adaptive Server for each database defined by the DATABASES configuration parameter.

Note With this step, the RCM completes the reversal of replication flow in the environment.

---

8 The RCM disconnects DSS users from the standby Adaptive Server. Typically, DSS users can be off-loaded to the standby Adaptive Server to execute read-only queries. You may decide to disconnect these users if a failover from the active to the standby Adaptive Server occurs. If you set the *DISCONNECT_STBY_USERS* configuration parameter, the RCM disconnects all users from the standby Adaptive Server before switching the users from the active Adaptive Server. The DSS users must wait to be reconnected when the active Adaptive Server is back online.

9 OpenSwitch switches end users from the active to the standby Adaptive Server. The RCM sets the server status to DOWN, switches the server connections from the active Adaptive Server to the standby Adaptive Server, and restarts all existing connections that were suspended at the active Adaptive Server.

How the RCM detects Adaptive Server failure

An Adaptive Server failure within the high-availability, warm-standby environment occurs if login requests or existing connections to the Adaptive Server fail. If the Adaptive Server fails, the OpenSwitch server passes the notification to the RCM.

1 The RCM attempts to connect to the Adaptive Server.

2 If the attempt fails, the RCM logs the server as DOWN.

If the attempt succeeds, the RCM determines if the requested database is available by monitoring database connections.

a If the requested database is listed in the *REQUIRED_DBS* configuration parameter, the RCM attempts to connect to the database. If the attempt fails, the server is considered down. If the attempt succeeds, the server is considered up.

b If the requested database is not in the list, the RCM considers only the status of the server and not the database when pinging the Adaptive Server. Because the server status is UP, the RCM does not begin the failover process.

This two-step process gives you finer control over failover. For example, you can prevent noncritical databases that become unavailable from starting the failover process.

Note Adaptive Server allows users to connect to the server even if the requested database is unavailable. End users receive an error message, but are still connected to the server. This means that the Adaptive Server does not notify the OpenSwitch server and, therefore, the RCM, when users attempt to connect to a database that is unavailable. However, the RCM is notified by the OpenSwitch server when existing connections fail because a database has become unavailable and the RCM can start the failover process.

How the RCM detects Replication Server failure

If the RCM cannot log in to the Replication Server, the RCM:

- Notifies the system administrator about a possible Replication Server failure and logs the failure in the system log.
- Waits a configurable interval of time to see if Replication Server recovers. This is required because network problems might prevent the connection.
- Continues with the failover process by marking the active Adaptive Server as DOWN and switching all users to the standby Adaptive Server.

Index

D

E

F

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W