Continuent.org Sequoia 2.10

continuent
Open. Always Available.

Continuent.org Sequoia 2.10 Management Guide

Document issue 2.1

Contents

1 About this document

This document comprises the Management Guide for Continuent.org Sequoia 2.10. It is organized as follows:

- chapter *2 Using the Command Line Console (CLC)* on page 2 discusses the basic concepts of cluster management using the Sequoia Command Line Console
- chapters 3-10 provide detailed instructions on the tasks and procedures involved in managing Sequoia with the Command Line Console
- Chapter *11 Command Line Console reference* on page 52 describes the commands of the Command Line Console and the arguments that they can receive.

1.1 Typographic conventions

This document uses the following formatting conventions:

Notation	Explanation
Italics	Indicates a reference to another document.
Hyperlink	Indicates a hyperlink to a web page.
Blue italics	Indicates a linked cross-reference to another section in the document.
Bold	Indicates elements in a graphical user interface.
`Courier`	Indicates code, including command line input and/or output.

2 Using the Command Line Console (CLC)

The Sequoia installation includes an integrated text-based management application that provides full control of all Sequoia features. The management application, called the Command Line Console (CLC) does not require the installation of any additional software. By default, the CLC is installed on the controller node.

With the CLC you can:

- administer the Sequoia middleware
- issue regular SQL statements against the cluster just as with a normal RDBMS client tool (see *11.2.10 `sql client` command* on page 55 and *11.6 `sql client` mode commands* on page 65).

The CLC also allows you to automate the cluster administration tasks by scripting the required CLC commands to a file and then feeding this file to the console (see 10.1 Command Line Console start-up options on page 43.)

The CLC is based on JMX technologies: the console is a JMX client based on the standard RMI connector for JMX.

Note

When you need to specify the controller IP address and port number in a CLC command, use the JMX/RMI port number, which is 1090 by default. The JMX port number is configured in the controller configuration file `controller.xml`.

The command line console has a Linux-style command line editing and history that allows you to use the arrow keys to recall previous commands and to edit commands. After entering a command, press the '`return`' key to execute it.

In CLC:

- all commands are lowercase, and commands and arguments are separated by spaces
- a response can be made up of a single line, or of multiple lines.

Note

The instructions in the following chapters assume that you have already logged on to the controller that you want to manage using SSH and started the CLC.

2.1 Start the Command Line Console (CLC)

By default, the CLC is installed on the controller host. Log on to the controller using an SSH connection to run the CLC locally.

To start the CLC

1. Log on to the controller using an SSH connection.
   ```
   $ ssh <host>
   ```
2. Move to the Sequoia installation directory.
   ```
   $ cd <install dir>
   ```
3. Start the CLC.

   To start the CLC in Linux:
   ```
   $ bin/console.sh
   ```
   To start the CLC in Windows:
   ```
   > bin\console.bat
   ```

Tip

If you are, for example, a Windows user and cannot use SSH to the controller, note that you can use the `-i` start-up option (see *Command Line Console start-up options*.)

2.2 CLC command menus and navigation commands

The Command Line Console has three different modes, that is, command menus. The set of available commands depends on which level of the console you are in.

The three console modes are:

- `main` - available when you start the Command Line Console
- `admin` - can be accessed from main menu
- `sql client` - can be accessed from the main menu.

To navigate between the different modes/menus, you can use the following commands:

- `admin <virtual database name>` - starts `admin` mode
- `sql client <Sequoia url>` - starts `sql client` mode
- `quit` - returns to the previous level in the console: for example, if you execute the `quit` command when in `admin` mode, you return to the main menu. If you execute this command in the main menu, it closes the console connection.
- `help` - displays the commands available for the current menu together with the command descriptions
- `history [<commandIndex>]` - displays the recently used commands for this mode. You can use the `commandIndex` option to invoke a command identified by its index number in the command history.

Note

When you go to `admin` mode, the CLC will prompt you for the administrator username and password and verify that they correspond to the administrator login in the virtual database configuration file. When you go to `sql client` mode, you will be prompted for the virtual username and password.

The `admin` mode has the following two additional command sets:

- expert administrator commands - see *`expert admin` commands*
- debugging commands - see *`debug` commands*.

Thus, the following two commands are only available in `admin` mode:

- `expert {on | off}` - enables or disables the use of the expert administrator command set
- `debug {on | off}` - enables or disables the use of the debugging command set.

2.3 Command syntax notation

In the examples shown in this document lines beginning with ">" are commands executed using CLC, and all other lines are responses:

- angle brackets "<>"indicate that you must enter a value for a given parameter
- brackets "`[]`" indicate that items between them are optional.
- ellipsis "`(...)`" indicates parameters that can be repeated several times on a command line
- braces "`{}`" indicate that items included between them are choices. These are separated by a pipe "`|`". The user may select only one of them.

2.4 Command Line Console (CLC) user privileges

When you execute certain commands in CLC, it prompts you for a username and password combination that you specified in the virtual database configuration file during Sequoia installation.

- To go to the `admin` mode of the CLC, you must enter the administrator login and password. See *2.2 CLC command menus and navigation commands* on page 3.
- When you execute the `restore backend` or `backup` commands, the CLC prompts you for the database username.
- When you go to `sql client` mode, you will be prompted for the virtual username and password.

Related topics

See *10 Sequoia user management* on page 50 for limitations and instructions related to changing the Sequoia user privileges.

See also section *Configuring Sequoia usernames and passwords* in *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

3 Controller management

The following sections provide detailed instructions about controller management tasks.

The controller management instructions assume that you have already started both the controller that you want to manage and the Command Line Console (see section *2.1 Start the Command Line Console (CLC)* on page 3.)

3.1 Start a controller

The correct way to start a controller in order to ensure cluster consistency depends on the cluster configuration and the current situation.

Refer to the following instructions under *7 Cluster maintenance* on page 23 if you are starting a controller node after a clean shutdown:

- *7.2.2 Restart a collocated controller node after maintenance* on page 26
- *7.3.2 Restart a dedicated controller node after maintenance* on page 29
- *7.4.2 Restart both controller nodes after maintenance* on page 32.

Refer to the following instructions under *8 Failure management* on page 36 if you are starting a controller after a controller failure:

- follow the instructions in *8.1 Recover from a controller node failure* on page 36 if one of the controllers has failed but another controller remains operational
- if all controllers have failed, you must activate the virtual database as instructed in sections *Activate the database backend(s) of the first controller* and *Activate the database backend(s) of the second controller* of *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

The Sequoia controller is run as a daemon. You can also run it in the foreground by using the `nobg` option of the `controller.sh` script. See *3.1.1 Controller start-up options* on page 7.

To start a controller

1. Log on to the controller using an SSH connection.
   ```
   > ssh <host>
   ```
2. Move to the Sequoia installation directory and start the controller.
   ```
   > cd <install dir>
   > bin/controller.sh [-f <filename>]
   ```
   To start a controller in windows:
   ```
   > bin/controller.bat [-f <filename>]
   ```

3.1.1 Controller start-up options

You can use the following options when launching the controller with the `controller.sh` script.

Option	Description
`nobg`	Specifies that the controller is run on the foreground (not as a daemon). This option must come before possible other controller start-up options.
`r,--rmi`	Optional JMX server RMI-Adaptor port number (the default port is 1090). Also enables JMX.
`-j,--jmx`	Optional JMX server HTTP-Adaptor port number (the default port is 8090). Also enables JMX.
`-f,--file`	The optional configuration file to initialize the controller.
`-h,--help`	Displays usage information.
`-i,--ip`	Optional IP address (the default IP address is 192.168.0.68)
`-p,--port`	Specifies the port the controller is listening on (the default port is 25322)
`-v,--version`	Displays version information.

3.2 Shut down a controller

The correct way to shut down a controller in order to ensure cluster consistency depends on the cluster configuration.

- Refer to the following instructions under *Cluster maintenance*:
  - *7.2.1 Shut down a collocated controller node for maintenance* on page 25
  - *7.3.1 Shut down a dedicated controller node for maintenance* on page 27
  - *7.4.1 Shut down both controller nodes for maintenance* on page 30.

3.3 Show the controller configuration

To show the contents of the controller configuration file `controller.xml`, execute the `show controller config` command.

3.4 Save the controller configuration

To save the current controller configuration as an XML file, execute the `save configuration` command.

```
save configuration <filename>
```

4 Virtual database management

The following sections provide instructions on general virtual database management tasks.

The virtual database management instructions assume that you have already started both the controller that you want to manage and the Command Line Console (see section *2.1 Start the Command Line Console (CLC)* on page 3.)

4.1 Load a virtual database configuration file

To load a new virtual database configuration file to the controller, for example, when activating a new cluster, execute the `load virtualdatabase configuration` command.

```
load virtualdatabase configuration <virtual database config
file>
```

Tip

You can use the *`show virtualdatabases` command* to check that the virtual database was loaded correctly.

4.2 Show the virtual databases hosted by a controller

To list the virtual databases hosted by the current controller, execute the `show virtualdatabases` command. If a virtual database is not shown by the `show virtualdatabases` command, this means that the virtual database has not been loaded into the controller in question.

Example

In this example the controller hosts one virtual database myDB_1.

```
> show virtualdatabases
myDB_1
```

4.3 Show the controllers hosting a virtual database

To show the controllers hosting a specific virtual database

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   ```
2. Enter the administrator username and password.
   ```
   > <admin username>
   > <admin password>
   ```
3. Execute the `show controllers` command.
   ```
   > show controllers
   ```

Example

```
> admin vdb_test_install
> username
> password
Ready to administrate virtual database vdb_test_install
> show controllers
vdb_test_install is hosted by 2 controller(s):
       192.168.0.58:1090
       192.168.0.59:1090
```

4.4 Shut down a virtual database

If you shut down a virtual database, it is no longer accessible on the selected controller. Note that client applications can still access the virtual database on other active controller(s).

To shut down a virtual database, execute the `shutdown virtualdatabase` command.

```
shutdown virtualdatabase <virtual database name> [<mode>]
```

Tip

You can retrieve a list of the virtual databases that are loaded into the current controller with the *`show virtualdatabases` command*. Similarly, after you shut down a virtual database, it is no longer shown when you execute the `show virtualdatabases` command.

4.5 Show the virtual database configuration

To show the contents of the virtual database configuration file, execute the `show virtualdatabase config` command.

To show the virtual database configuration

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   ```
2. Enter the administrator username and password.
   ```
   > <admin username>
   > <admin password>
   ```
3. Show the virtual database configuration
   ```
   > show virtualdatabase config
   ```

4.6 Save the virtual database configuration

To save the current virtual database configuration under a different filename, use the `show virtualdatabase config` command to first output the contents of the virtual database configuration file and then copy-paste the output to a text editor.

You can also use a `sed` script to automate this task as shown in the example below.

Example

This example shows how you can use the `-f` option of the Command Line Console start-up script to automate cluster management and run a file that includes the required CLC commands (see *11.1 Command Line Console start-up options* on page 52.)

Here, a `sed` script is used to save the command line output of the `show virtual database config` command as a virtual database configuration file.

```
# bin/console.sh -f save-tmp.txt 2>/dev/null | sed '1,6 d;$d' >
/tmp/new-vdb.xml
```

The contents of the `save-tmp.txt` file are shown below. The `save-tmp.txt` file must include the CLC commands and administrator username and passwords required to show the virtual database configuration.

```
admin <virtual database name>
<admin username>
<admin password>
show virtualdatabase config
```

5 Backend management

The following sections instruct how to perform database backend management tasks.

A *backend* is the Sequoia object that is used to manage an underlying database server. In other words, *backend* is used to refer to a view of the underlying database, which belongs to a virtual database. The `<DatabaseBackend/>` definition in a virtual database configuration file defines which backends are associated with which database servers.

The instructions assume that you have already started both the controller that you want to manage and the Command Line Console (see section *2.1 Start the Command Line Console (CLC)* on page 3.)

5.1 Show the backends hosted by the controller

To list the backends for the current controller, execute the `show backends` command.

To list the backends for the current controller

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Show the backends.
   ```
   > show backends
   ```

Example

In this example the controller hosts two backends `db_server_1` and `db_server_2`.

```
> admin vdb_test_install
> username
> password
> show backends
db_server_1
db_server_2
```

5.2 Show backend properties

To output information about the status and properties of a backend, execute the `show backend` command.

To show the backend properties

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Show the backend properties.
   ```
   show backend {<backend name> | *}
   ```

Example

In this example the controller hosts two backends, `db_server_1` and `db_server_2`. The properties of the two backends are shown below.

```
myDB(admin) > show backend *
+--------------------------+-----------------------------------+
| Backend Name             | db_server_1                       |
| Driver                   | org.hsqldb.jdbcDriver             |
| URL                      | jdbc:hsqldb:hsql://localhost:9001 |
| Active transactions      | 0                                 |
| Pending Requests         | 0                                 |
| Read Enabled             | true                              |
| Write Enabled            | true                              |
| Is Initialized           | true                              |
| Static Schema            | false                             |
| Connection Managers      | 2                                 |
| Total Active Connections | 25                                |
| Total Requests           | 10                                |
| Total Transactions       | 1                                 |
| Last known checkpoint    | <unknown>                         |
+--------------------------+-----------------------------------+
| Backend Name             | db_server_2                       |
| Driver                   | org.hsqldb.jdbcDriver             |
| URL                      | jdbc:hsqldb:hsql://localhost:9002 |
| Active transactions      | 0                                 |
| Pending Requests         | 0                                 |
| Read Enabled             | true                              |
| Write Enabled            | true                              |
| Is Initialized           | true                              |
| Static Schema            | false                             |
| Connection Managers      | 2                                 |
| Total Active Connections | 25                                |
| Total Requests           | 10                                |
| Total Transactions       | 1                                 |
| Last known checkpoint    | <unknown>                         |
+--------------------------+-----------------------------------+
```

5.3 Changing the backend state

The possible states of the backend are described in the following table.

Backend state	Description
enabled	The backend / underlying database server can execute read and write requests.
disabled	The backend / underlying database server cannot execute any requests.
disabling	The backend is being transferred to disabled state.
backuping	The database is being backed up to a dump file.
recovering	The backend / underlying database server is loading data from a database dump file.
replaying	The backend is being enabled and is synchronizing the underlying database by replaying requests from the recovery log.
unknown	Unknown backend state. The backend may be transferred to unknown state automatically after a restore or a recovery failure.

5.3.1 Enable a backend

You can enable a backend by executing the `enable` command if there is a checkpoint in the recovery log that can be used to synchronize the underlying database server.

Note that if you are enabling a failed backend that has no clean shutdown point, you must first restore a database dump to it to ensure database consistency (see *8.2 Recover from a database server node failure* on page 37.)

To enable a backend

1. Go to `admin` mode.
```
> admin <virtual database name>
> <admin username>
> <admin password>
```
2. Enable the backend.
```
> enable {<backend name> | *}
```

5.3.2 Disable a backend

To disable a backend, execute the `disable` command. When you disable a backend, a checkpoint is recorded to the recovery log so that when the backend is again enabled, the underlying database is synchronized automatically by replaying the missing entries from the recovery log.

To disable a backend

1. Go to `admin` mode.
```
> admin <virtual database name>
> <admin username>
> <admin password>
```
2. Disable the backend.
```
> disable {<backend name> | *}
```

5.4 Clone a backend

By cloning a backend you can copy the configuration of an existing backend in the current virtual database to create a new backend definition.

After cloning the existing backend, you can activate the new backend by first restoring the database to it from a database dump and then enabling it.

To clone a backend

1. Go to `admin` mode and enable the use of the `expert admin` commands.
```
> admin <virtual database name>
> <admin username>
> <admin password>
> expert on
```
2. Clone the backend configuration.
```
> clone backend config <backend from> <backend to> <URL>
[driverPath=<value>]
```
3. Disable the use of the `expert admin` commands.
```
> expert off
```
4. Restore the database to the new backend from a database dump.
```
> restore backend <backend name> <dump name>
> <database username>
> <database password>
```
5. Enable the new backend.
```
> enable {<backend name> | *}
```

6 Database backup management

Backups play an important role in recovering from a hardware or disk failure of a cluster node. Thus, you should back up the database servers on a regular basis. In addition, remember to purge the recovery log to ensure that its size stays below the maximum limit.

We recommend the use of crontab to ensure the cluster is backed up regulary. For example, create a crontab file to:

1. back up the database backend(s) of `controller1`
2. purge the recovery log of `controller1`
3. transfer the database dump to `controller2`
4. purge the recovery log of `controller2`.

See the following sections for more details on the backup management commands. The following instructions assume that you have already started both the controller that you want to manage and the Command Line Console (see section *2.1 Start the Command Line Console (CLC)* on page 3.)

6.1 Back up a database server

To back up a database server and create a database dump of its database, execute the `backup` command.

When you back up a database server:

- to ensure database consistency, the backend is disabled to prevent incoming client requests to be executed on the database server that is being backed up
- a checkpoint is recorded in the recovery log
- the database server performs the actual backup
- the backend is re-enabled automatically and the missing client requests are replayed starting from the previously recorded checkpoint to synchronize the underlying database.

Note

For the `transfer dump` command to work correctly, you must use an identical directory structure for storing dumps in all controllers.

Warning

If you execute the `backup` command when there is only one enabled backend available, the CLC warns you that the backup operation will also bring the remaining backend to disabled state for the time of the backup. Thus, the whole cluster will stop serving requests.

To back up a database server

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Create a database backup.
   ```
   > backup <backend name> <dump name> <backuper> <path>
   ```
3. Enter the database username and password.
   ```
   > <database username>
   > <database password>
   ```

Example

In this example, the database associated with a backend named `backend1` is backed up using the PostgreSQL backuper. The name of the created database dump is `myDump`.

```
> admin DB1
> admin
> ******
Ready to administrate virtual database DB1
> backup backend1 myDump postgresql /dumps
> realuser
> ******
Backup backend backend1 in dump file myDump
```

6.2 Show the available database dumps

To list the available database dumps, execute the `show dumps` command.

To show the available database dumps

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `show dumps` command.
   ```
   > show dumps
   ```

Example

In this example the virtual database has only one database dump, `dump1`.

```
> admin vdb_test_install
> admin
> ******
Ready to administrate virtual database vdb_test_install
> show dumps
+-------+------------+--------------+------+------+------ ---+--------+
| Name  | Checkpoint | Format       | Path | Date | Backend  | Tables |
+-------+------------+--------------+------+------+----------+--------+
| dump1 | Initial_...| PostgreSQL...| /tmp | Tue..| backend_1| *      |
+-------+------------+--------------+------+------+----------+--------+
```

6.3 Show the available backupers

To output a list of the currently available backupers that can be used to create a database dump, execute the `show backupers` command.

To show the backupers

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `show backupers` command.
   ```
   > show backupers
   ```

6.4 Restore a database server from a database dump

To restore the database from a database dump file, execute the `restore backend` command.

To restore a database server from a database dump

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `restore backend` command.
   ```
   > restore backend <backend name> <dump name>
   ```
3. Enter the database username and password.
   ```
   > <database username>
   > <database password>
   ```

6.5 Delete a database dump

By default, executing the `delete dump` command removes both the dump entry and the actual physical backup file from the file system. To remove just the dump entry on the current controller, use the `keepfile` option.

To delete an existing database dump

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `delete dump` command.
   ```
   > delete dump <dump name> [keepfile]
   ```

6.6 Transfer a database dump to another controller

With the `transfer dump` command you can transfer a database dump from one controller to another so that it becomes available on another (remote) controller:

- by default, the data of the dump file is copied to the other controller machine
- with the `nocopy` option you can specify that the controllers share the dump through a network file system: the dump is made available by sending the dump name and metadata (path, type, date) to the remote controller.

To transfer a database dump

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `transfer dump` command.
   ```
   > transfer dump <dump name> <controller IP>:<jmx port>
   [nocopy]
   ```

Note

The remote controller to which you transfer the database dump must host the same virtual database for the dump transfer to be successful.

Note

For the `transfer dump` command to work correctly, you must use an identical directory structure for storing dumps in all controllers.

6.7 Purge the recovery log

To ensure that the size of the recovery log stays below the maximum limit, perform backups and purge the recovery log on a regular basis.

To purge the recovery log

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Execute the `purge log` command.
   ```
   > purge log <dump name>
   ```

6.8 Copy and restore the recovery log to another controller

The recovery log is defined per virtual database and per controller. You can synchronize the recovery log of a remote controller by restoring the recovery log of the current controller to it.

To restore a recovery log to another controller

1. Go to `admin` mode and enable the use of the `expert admin` commands.

```
> admin <virtual database name>
> <admin username>
> <admin password>
> expert on
```

2. Execute the `restore log` command.

```
> restore log <dump name> <controller IP>:<jmx port>
```

7 Cluster maintenance

The following sections provide instructions on how to bring down a cluster node for maintenance, making sure that the database consistency is not compromised. Note that the way this is done depends on:

- the configuration you are using (dedicated/collocated controller configuration)
- whether you need to bring down the whole cluster (all controllers) or whether one of the controllers will remain operational.

Before performing maintenance tasks on a controller node, you must shut down the virtual database(s) and the controller. When the maintenance tasks have been completed, restart the controller as instructed below.

The following instructions show the correct maintenance procedure for a cluster with two controllers.

7.1 Guidelines to prevent maintenance failures

Before performing maintenance on the cluster, make sure that:

- for each configured virtual database, you have at least one valid database dump from which to recover in case of a failure
- the dump is available on both controller nodes hosting the virtual database.

To check that there are database dumps available for all virtual databases hosted on the machine to be shut down

1. Start the CLC.
   ```
   > <install dir>/bin/console.sh
   ```
2. Show the virtual databases hosted by this controller.
   ```
   > show virtualdatabases
   ```
3. Check that there is a valid database dump available for each virtual database hosted by this controller.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   > show dumps
   ```
   - If there is no dump available, create it with the `backup` command (see *6.1 Back up a database server* on page 17).
   - If there is a dump available, move to step 4.

4. Check which other controllers host the same virtual database(s) as this controller.

```
> admin <virtual database name>
> <admin username>
> <admin password>
> <show controllers>
```

5. Make sure that the same, valid dump exists on the other controller(s) hosting the same virtual database(s).

```
> <install dir>/bin/console.sh
> admin <virtual database name>
> <admin username>
> <admin password>
> show dumps
```

   If not, transfer a valid dump from one controller to another using the `transfer dump` command (see *6.6 Transfer a database dump to another controller* on page 20).

7.2 Perform maintenance on a collocated controller node

In this maintenance scenario, a machine hosting both a controller and a database server (a collocated controller node) is removed from the cluster.

After it has been removed from the cluster:

- the controller node is no longer operational
- the node can be rebooted
- any maintenance operations that do not affect the Sequoia configuration or the database contents can be performed on the node.

Warning

Bringing down a controller node as instructed here means that there will no longer be high availability for the cluster. If a failure occurs on the other controller node, the whole cluster will become unoperational. In such a case, you must reactivate the virtual database as instructed in sections *Activate the database backend(s) of the first controller* and *Activate the database backend(s) of the second controller* of *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

Note

The following instructions only apply if you are bringing one collocated controller down at a time. If you need to bring both controllers down for maintenance, you must shut them down and bring them up in the respective order: shut down `controller1`, shut down `controller2`; restart `controller2`, restart `controller1`. See *Perform maintenance on the whole cluster (shut down all controllers)*. This allows the automatic synchronization of the recovery log.

7.2.1 Shut down a collocated controller node for maintenance

To perform a clean shutdown of a collocated controller node, follow the instructions below:

- you must shut down the controller as instructed to allow automatic resynchronization of the database at restart
- if you receive an error while following the instructions, and are unable to bring the node down, force the shutdown of the virtual database (use value `3` for the `mode` option of the `shutdown virtualdatabase` command).

Warning

Forcing the shutdown of the virtual database can compromise database consistency. Before you re-enable the backends hosted by this controller, you must first restore the database from a valid database dump as instructed in *8.1 Recover from a controller node failure* on page 36.

To shut down a collocated controller node

1. Follow the guidelines in *7.1 Guidelines to prevent maintenance failures* on page 23.
2. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
3. Shut down the virtual database so that it is no longer accessible on this controller. The CLC prompts you for the administrator username and password.
   ```
   > shutdown virtualdatabase <virtual database name> [<mode>]
   > <admin username>
   > <admin password>
   ```

4. Shut down the controller.
   ```
   > shutdown
   ```
5. Quit the CLC.
   ```
   > quit
   ```

7.2.2 Restart a collocated controller node after maintenance

You can restart a collocated controller node as instructed below if it was shut down properly (see *7.2.1 Shut down a collocated controller node for maintenance* on page 25). The database is synchronized automatically by replaying all missing client update requests starting from the checkpoint that was recorded during shutdown.

Warning

If you need to restart the controller node after a failure, you must follow the instructions in *8.1 Recover from a controller node failure* on page 36.

To restart a collocated controller node after maintenance

1. Start the controller.
   ```
   > <install dir>/bin/controller.sh &
   ```
2. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
3. Load the virtual database configuration file.
   ```
   > load virtualdatabase configuration <virtual database
   config file>
   ```
4. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
5. Enable the backend(s).
   ```
   > enable *
   ```
6. Quit admin mode.
   ```
   > quit
   ```
   Quit the console.
   ```
   > quit
   ```

7.3 Perform maintenance on a dedicated controller node

If you need to shut down one dedicated controller node, that is, a machine that hosts only the Sequoia controller and no database server(s), you can keep the database servers in operation during the maintenance by transferring the backends used to control them temporarily under the control of another controller node that remains available.

After it has been removed from the cluster:

- the controller node is no longer operational
- the node can be rebooted
- any maintenance operations that do not affect the Sequoia configuration or the database contents can be performed on the node.

Warning

Bringing down a controller node as instructed here means that there will no longer be high availability for the cluster. If a failure occurs on the other controller node, the whole cluster will become unoperational. In such a case, you must reactivate the virtual database as instructed in sections *Activate the database backend(s) of the first controller* and *Activate the database backend(s) of the second controller* of *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

Note

The following instructions only apply if you are bringing one dedicated controller down at a time. If you need to bring both controllers down for maintenance, you must shut them down and bring them up in the respective order: shut down `controller1`, shut down `controller2`; restart `controller2`, restart `controller1`. See *Perform maintenance on the whole cluster (shut down all controllers)*. This allows the automatic synchronization of the recovery log.

7.3.1 Shut down a dedicated controller node for maintenance

To perform a clean shutdown of a dedicated controller node, follow the instructions below:

- you must shut down the controller as instructed to ensure database consistency
- if you receive an error while following the instructions, and are unable to bring the node down, force the shutdown of the virtual database (use value `3` for the `mode` option of the `shutdown virtualdatabase` command).

Warning

Forcing the shutdown of the virtual database can compromise database consistency. Before you re-enable the backends hosted by this controller, you must first restore the database from a valid database dump as instructed in *8.1 Recover from a controller node failure* on page 36.

To shut down a dedicated controller node

1. Follow the guidelines in *7.1 Guidelines to prevent maintenance failures* on page 23.
2. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
3. Go to `admin` mode and enable the use of the `expert admin` commands.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   > expert on
   ```
4. Tranfer the backend(s) to another active controller. By transferring the database backend(s) you remove the `<DatabaseBackend/>` definitions from the runtime configuration of the virtual database on the current controller.
   ```
   > transfer backend <backend name> <controller ip>:<jmx port>
   ```
5. Disable the use of the `expert admin` commands.
   ```
   > expert off
   ```
6. Save the modified virtual database configuration by copying it from the output of the `show virtualdatabase config` command to a text editor. You need to save the empty virtual database configuration file (that is, a virtual database with no backends) so that you can reload it to the controller after the maintenance.
   ```
   > show virtualdatabase config
   ```
7. Quit `admin` mode and return to the main menu.
   ```
   > quit
   ```
8. Shut down the virtual database(s) so that it is no longer accessible on this controller. The CLC prompts you for the `admin` username and password.
   ```
   > shutdown virtualdatabase <virtual database name> [<mode>]
   > <admin username>
   > <admin password>
   ```
9. Repeat steps 3-8 for all virtual databases hosted by this controller.

10. Shut down the controller and quit the CLC.
    ```
    > shutdown
    ```
11. Quit the CLC.
    ```
    > quit
    ```

7.3.2 Restart a dedicated controller node after maintenance

You can restart a dedicated controller node as instructed below if it was shut down properly (see *7.3.1 Shut down a dedicated controller node for maintenance* on page 27).

Warning

If you need to restart the controller node after a failure, you must follow the instructions in *8.1 Recover from a controller node failure* on page 36.

After controller maintenance, you must load an empty virtual database configuration file to the controller in order to be able to transfer the database servers back to it. Because each `<DatabaseBackend/>` definition in a virtual database configuration must be unique, the `transfer backend` command will fail if you try to add an existing database server to the same virtual database configuration.

To restart a dedicated controller node

1. Restart the controller.
   ```
   > <install dir>/bin/controller.sh &
   ```
2. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
3. Load the empty virtual database configuration file that you created in *7.3.1 Shut down a dedicated controller node for maintenance* on page 27 (step 6) to the controller.
   ```
   > load virtual database configuration <virtual database
   config file>
   ```
4. Connect to the other controller where you transferred the backend(s).
5. Start the CLC.

6. Go to `admin` mode and enable the use of the `expert admin` commands.

```
> admin <virtual database name>
> <admin username>
> <admin password>
> expert on
```

7. Transfer the backend(s) back to the controller where you performed the maintenance.

```
> transfer backend <backend name> <controller ip>:<jmx port>
```

The cluster is now again fully operational, with high availability.

7.4 Perform maintenance on the whole cluster (shut down all controllers)

In this maintenance scenario, both controllers are shut down, thus making the whole cluster unoperational for the time of the maintenance.

After they have been removed from the cluster:

- the controller nodes are no longer operational
- the nodes can be rebooted
- any maintenance operations that do not affect the Sequoia configuration or the database contents can be performed on the nodes.

To bring all controllers down for maintenance, you must shut them down and bring them up in the respective order:

1. shut down `controller1`
2. shut down `controller2`
3. restart `controller2`
4. restart `controller1`.

This allows the automatic synchronization of the recovery log and ensures database consistency.

7.4.1 Shut down both controller nodes for maintenance

To perform a clean shutdown of the controller nodes, follow the instructions below:

- you must shut down the controllers as instructed to ensure database consistency
- if you receive an error while following the instructions, and are unable to bring the node down, force the shutdown of the virtual database (use value `3` for the `mode` option of the `shutdown virtualdatabase` command).

Warning

Forcing the shutdown of the virtual database can compromise database consistency. Before you re-enable the backends hosted by this controller, you must first restore the database from a valid database dump as instructed in *8.1 Recover from a controller node failure* on page 36.

If you force the shutdown of the virtual database when only one controller is running, the procedure *8.1 Recover from a controller node failure* on page 36 cannot be applied. You must initialize the whole virtual database as instructed in sections *Activate the database backend(s) of the first controller* and *Activate the database backend(s) of the second controller* of *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

To shut down both controller nodes

1. Follow the guidelines in *7.1 Guidelines to prevent maintenance failures* on page 23.
2. On the first controller node (`controller1`), start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
3. Show the virtual databases hosted by this controller.
   ```
   > show virtualdatabases
   ```
4. Shut down the virtual database so that it is no longer accessible on this controller. The CLC prompts you for the administrator username and password.
   ```
   > shutdown virtualdatabase <virtual database name> [<mode>]
   > <admin username>
   > <admin password>
   ```
5. If the controller hosts several virtual databases, repeat this step for the other virtual databases that were listed by the `show virtualdatabases` command.
6. Shut down the controller (`controller1`).
   ```
   > shutdown
   ```
7. Quit the CLC.
   ```
   > quit
   ```

8. Repeat steps 1-6 on the other controller node(s).

   The database servers can now be stopped. Note that no data updates must be made while the backends and thus the database servers are not under the control of the controllers. This will break the consistency of the cluster and prevent making a clear restart as described in *Restart both controller nodes after maintenance*.

7.4.2 Restart both controller nodes after maintenance

To restart both controller nodes

1. Make sure that the database servers are up and running.
2. Start `controller2`, that is, the controller which you shut down last.
   ```
   > <install dir>/bin/controller.sh
   ```
3. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
4. Load the virtual database configuration file to `controller2`.
   ```
   > load virtual database configuration <virtual database
   config file>
   ```
5. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
6. Enable the backend(s) of `controller2`.
   ```
   > enable *
   ```
7. Exit `admin` mode.
   ```
   > quit
   ```
8. Repeat steps 4 - 7 for all other virtual databases hosted on `controller2`.
9. Quit the console.
   ```
   > quit
   ```

10 Start `controller1`.

11. Start the CLC.
12. Load the virtual database configuration file to `controller1`.
    ```
    > load virtual database configuration <virtual database
    config file>
    ```

13. Go to `admin` mode.
    ```
    > admin <virtual database name>
    > <admin username>
    > <admin password>
    ```
14. Enable the backend(s) of `controller1`.
    ```
    > enable *
    ```
15. Exit `admin` mode.
    ```
    > quit
    ```
16. Repeat steps 12 - 15 for all other virtual databases hosted on `controller1`.

7.5 Perform maintenance on a database server node

If you need to shut down a machine that only hosts a database server and no controller:

- disable the backend first as instructed below
- if the machine in question hosts backends for several virtual databases, you must disable all backends for all virtual databases on this machine.

When you disable the backend(s), this means that the database server is removed from the cluster control. Disabling a backend also adds a checkpoint to the recovery log. When you re-enable the backend after the maintenance, the database is synchronized automatically by replaying all missing client update requests starting from the checkpoint that was recorded during disabling.

Note

To shut down a machine that hosts both a controller and a database server, follow the instructions in *7.2 Perform maintenance on a collocated controller node* on page 24.

7.5.1 Remove a database server from the cluster

To remove a database server from the cluster

1. On the controller that controls the database server node that you want to remove from the cluster, start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
2. List the virtual database(s) hosted by this controller.
   ```
   > show virtualdatabases
   ```
3. To list the backends belonging to a virtual database, execute the `show backends` command.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   > show backends
   ```
4. Repeat step 3 for all virtual databases whose backend names you want to check (first quit `admin` mode for the first virtual database.)
5. Disable the backends that are used to control the database server that you want to remove from the cluster.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   > disable {<backend name> | *}
   ```
6. Repeat step 5 for all virtual databases / backends of the database server node.

7.5.2 Add a database server node to the cluster

To add a database server node to a cluster, that is, under the control of a controller, follow the instructions below.

Note

These instructions only apply for a node that was properly removed from the cluster by disabling the corresponding backend(s) and that has a clean checkpoint that can be used to replay the missing client update requests. To enable a failed backend for which there is no clean shutdown point recorded in the recovery log, follow the instructions in *8.2 Recover from a database server node failure* on page 37.

To add a database server node to the cluster

1. Start the CLC (see *2.1 Start the Command Line Console (CLC)* on page 3.)
   ```
   > <install dir>/bin/console.sh
   ```
2. Enable the backend used to control the database server.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   > enable {<backend name> | *}
   ```
3. Repeat step 2 for all virtual databases / backends of this database server node.

8 Failure management

Follow the instructions in the following sections to recover from a controller node or database node failure or to obtain information for debugging purposes.

8.1 Recover from a controller node failure

In a situation where one of the controllers has failed but the other controller is still operational, follow the instructions below to enable the backends.

If both controllers have failed, you must activate the virtual database as instructed in sections *Activate the database backend(s) of the first controller* and *Activate the database backend(s) of the second controller* of *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

To recover from a controller failure when only one controller has failed

1. Start the failed controller. See *3.1 Start a controller* on page 6.
2. Start the CLC. See *2.1 Start the Command Line Console (CLC)* on page 3.
3. Load the virtual database configuration file.
   ```
   > load virtual database configuration <virtual database
   config file>
   ```
4. Connect to the second controller and go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
5. Check that the second controller has a database dump that you can use to restore the node(s).
   ```
   > show dumps
   ```
6. Copy the database dump to the failed controller.
   ```
   > transfer dump <dump name> <controller ip>:<jmx port>
   [nocopy]
   ```
7. Go to `admin` mode and enable the use of the `expert admin` commands.
   ```
   > expert on
   ```
8. Synchronize the recovery log of the failed controller.
   ```
   > restore log <dump name> <controller ip>:<jmx port>
   ```
9. Disable the use of the `expert admin` commands.
   ```
   > expert off
   ```

10. On the failed controller, go to `admin` mode.
```
> admin <virtual database name>
> <admin username>
> <admin password>
```
11. Restore the backend of the failed controller.
```
> restore backend <backend name> <dump name>
```
12. Enter the database username and password.
```
> <database username>
> <database password>
```
13. Enable the backend of the failed controller. Enabling the backend synchronizes the underlying database with the other cluster nodes.
```
> enable {<backend name> | *}
```
14. If you are using a configuration where there is more than one backend per controller, repeat steps 11-13 for the other backend(s) to activate it.

8.2 Recover from a database server node failure

To activate a database server node after a database server failure, you must first restore a dump file to it and then enable it. If you do not have a dump file that you can use to restore the database, create one as instructed in *6.1 Back up a database server* on page 17.

To activate a failed database server node

1. Go to `admin` mode on the controller that hosts the backend you want to enable.
```
> admin <virtual database name>
> <admin username>
> <admin password>
```
2. Show the available database dump files.
```
> show dumps
```
3. Use a database dump to restore the backend that you want to enable.
```
> restore backend <backend name> <dump name>
```
4. Enter the database username and password.
```
> <database username>
> <database password>
```
5. Enable the backend. Enabling the backend synchronizes the underlying database with the other cluster nodes.
```
> enable {<backend name> |*}
```

Note

If the `enable` command fails, this can be caused by a missing checkpoint in the recovery log. If there is no clean checkpoint in the recovery log, the database server cannot be synchronized by replaying the recovery log entries. In such a case, you must first synchronize the recovery log. See *11.4.6 `restore log` command* on page 62 for more details.

Example

In this example a database is restored using the latest available database dump, `myDump`. The associated backend, named `backend1`, is then enabled.

```
> admin DB1
> admin
> ******
Ready to administrate virtual database DB1
> show dumps
+--------+--------------+------------+------+------+----------+--------+
| Name   | Checkpoint   | Format     | Path | Date | Backend  | Tables |
+--------+--------------+------------+------+------+----------+--------+
| myDump | disable_...  | PostgreSQL| .    | Fri..| backend1 | *      |
|init... | Init_empty...| PostgreSQL| /tmp | Thu..| backend1 | *      |
+--------+--------------+------------+------+------+----------+--------+

> restore backend backend1 myDump
> realuser
Restoring backend backend1 with dump myDump
> enable backend1
Enabling backend backend1 from its last known checkpoint
```

8.3 Debug Sequoia

Sequoia has a set of debugging commands, the use of which is described in the following sections. Note that these commands must first be enabled from the `admin` mode using the `debug` command (see *2.2 CLC command menus and navigation commands* on page 3.)

8.3.1 Show the contents of the recovery log

To show the contents of the recovery log for debugging purposes, for example, execute the `dump recoverylog` command. The contents of the recovery log are displayed one screen at a time, allowing you to scroll the output.

To show the contents of the recovery log

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. Show the contents of the recovery log.
   ```
   > dump recoverylog {[indexes] | [<min> <max>]}
   ```
4. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

8.3.2 Show the database schema of a database server

To show the current database schema of a given database server, execute the `dump backend schema` command.

You can use the `dump backend schema` command to output the following:

- a list of tables and the locks on these tables
- table columns.

To show the database schema

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. Show the database schema.
   ```
   > dump backend schema <backend name> [table name] [/columns]
   [/locks]
   ```
4. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

Example

```
> admin myDB
> admin
>
> debug on
> dump backend schema localhost
PUBLIC.DOCUMENT
PUBLIC.PRODUCT
PUBLIC.ADDRESS
PUBLIC.PPOSITION
```

Example

```
> admin myDB
> admin
>
> debug on
> dump backend schema localhost PUBLIC.PRODUCT /columns /locks
PUBLIC.PRODUCT
    ID
    NAME
    COST
Lock owner transaction id: 112
```

8.3.3 Show information about pending requests and transactions

The request scheduler maintains an ordered list of the pending requests and transactions about to be executed. This allows you to retrieve the information about a request or transaction that may be blocking the controller with the `dump scheduler queues` and `dump request` commands:

- The `dump scheduler queues` command shows a list of all currently pending transactions, read requests, and write requests. It shows the size and contents of these pending queues as a list of IDs.
- The `dump request` command shows information about a specific request currently pending in the request scheduler queue.

To show information about the pending requests and transactions

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. List all pending requests and transactions.
   ```
   > dump scheduler queues
   ```
4. If you want to view more details about a specific pending request, execute the `dump request` command.
   ```
   > dump request <request id>
   ```
5. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

Example

```
> admin myDB
> admin
>
> debug on
> dump scheduler queues
Active transactions: 0
        Transaction id list:
Pending read requests: 1
        Read request id list: 6950
Pending write requests: 1
        Write request id list: 6955
```

Example

```
> admin myDB
> admin
>
> debug on
> dump request 6955
Request id: 6955
   query: UPDATE product SET cost=1 WHERE id=1
   login: user
   autocommit: true
   transaction id: 6955
   cacheable status: UNCACHEABLE
   isolation level: TRANSACTION_UNDEFINED
   start time: 1146129922441
   end time: 0
   timeout in seconds: 0
   locked tables:
      PRODUCT
   persistent connection id: 0
   client ip address: /127.0.0.1
```

8.3.4 Show the pending requests for a backend

To show the pending requests for a specific backend for debugging purposes, for example, execute the `dump queues` command.

To show the request queues

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. Show the request queues.
   ```
   > dump queues <backend name>
   ```
4. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

8.3.5 Show the request parsing cache configuration and entries

To show the configuration of the request parsing cache and list its entries (both requests that have been / currently are being parsed), execute the `dump parsing cache` command.

The request parsing cache:

- maintains a list of requests that have been or are currently being parsed
- allows the direct retrieval of the parsing result for a request that is to be executed several times, thus saving the time of a new unnecessary parsing.

When you execute the `dump parsing cache` command, the entries in the parsing cache are displayed in the following format.

```
<unique key>: <request>
```

Where:

- `unique key` is generally the request string
- `request` is a small dump of the request internals.

If the parsing cache dump contains more than 50 entries, the screen will be locked every 50 lines:

- to display the next 50 lines, press `<enter>`
- to quit the `dump parsing cache` command, press `<q>`.

To show the request parsing cache

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. Execute the `dump parsing cache` command.
   ```
   > dump parsing cache
   ```
4. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

Example

```
> admin myDB
> admin
>
> debug on
> dump parsing cache
Parsing cache:
        Granularity=TABLE
        Max number of entries=5000
        Background parsing=false
        Case sensitive=false

Cache entries: (unique key: request)

UPDATE product SET cost=885 WHERE id=35: Id=901, query=UPDATE product SET
cost=885 WHERE id=35, transaction id=899, persistent connection id=0

UPDATE product SET cost=308 WHERE id=8: Id=324, query=UPDATE product SET
cost=308 WHERE id=8, transaction id=322, persistent connection id=0

UPDATE product SET cost=125 WHERE id=25: Id=141, query=UPDATE product SET
cost=125 WHERE id=25, transaction id=139, persistent connection id=0

UPDATE product SET cost=25 WHERE id=25: Id=41, query=UPDATE product SET
cost=25 WHERE id=25, transaction id=39, persistent connection id=0

UPDATE product SET cost=242 WHERE id=42: Id=258, query=UPDATE product SET
cost=242 WHERE id=42, transaction id=256, persistent connection id=0

UPDATE product SET cost=194 WHERE id=44: Id=210, query=UPDATE product SET
cost=194 WHERE id=44, transaction id=208, persistent connection id=0

UPDATE product SET cost=458 WHERE id=8: Id=474, query=UPDATE product SET
cost=458 WHERE id=8, transaction id=472, persistent connection id=0

UPDATE product SET cost=94 WHERE id=44: Id=110, query=UPDATE product SET
cost=94 WHERE id=44, transaction id=108, persistent connection id=0

UPDATE product SET cost=870 WHERE id=20: Id=886, query=UPDATE product SET
cost=870 WHERE id=20, transaction id=884, persistent connection id=0

UPDATE product SET name='changed by testExecuteWithUpdate' WHERE id=0:
Id=1023, query=UPDATE product SET name='changed by testExecuteWithUpdate'
WHERE id=0, transaction id=1015, persistent connection id=0

Press enter to display next entries, 'q' to stop dump >
```

8.3.6 Show the request parsing results

You can use the `parse request` command to call the Sequoia request parser with a given SQL request and show the expected result of this request parsing. The parsing lists the modifications that would occur on the database if the request was executed.

Note

The execution of the `parse request` command has no effect on the database. The request is NOT executed, nor added in the parsing cache.

To show the request parsing results

1. Go to `admin` mode.
   ```
   > admin <virtual database name>
   > <admin username>
   > <admin password>
   ```
2. Enable the use of the `debug` command set.
   ```
   > debug on
   ```
3. Execute the `parse request` command.
   ```
   > parse request <sql string>
   ```
4. To disable the use of the `debug` command set, execute the following command.
   ```
   > debug off
   ```

Example

```
myDB(admin) > parse request "update product set cost=12.3 where id=0"
Request parsing results:
        Type=UPDATE
        Unique key=update product set cost=12.3 where id=0
        Locked tables:
                PRODUCT
        Table involved in write=PRODUCT
        Blocking=true
        Primary key=null!request.update.!
        Alters:
                AggregateList=false,
                DatabaseCatalog=false,
                DatabaseSchema=false,
                MetadataCache=false,
                QueryResultCache=true,
                Something=true,
                StoredProcedureList=false,
                UserDefinedTypes=false,
                Users=false
```

9 Logging management

You can control the logging configuration at runtime by modifying the logger configuration file `log4j.properties` in the `config` directory.

9.1 Show the logging configuration

To show the logging configuration, that is, the contents of the `log4j.properties` file, execute the `show logging config` command.

9.2 Change the logging configuration

To change the logging configuration

1. Edit the logging configuration file `log4j.properties` as instructed in sections *Enable/disable logging or change the logging level* and *Configure the log files to be created*.
2. Reload the logging configuration.
   ```
   > reload logging configuration
   ```

9.2.1 Enable/disable logging or change the logging level

To enable or disable logging, or change the logging level, edit the component's logger definition.

There are five logging levels available:

- `DEBUG` - logs all messages
- `INFO` - logs general level messages about system activities
- `WARN` - logs warnings that are usually generated as a response to faulty command line input
- `ERROR` - logs errors generated by the controller (including request execution failures)
- `FATAL` - logs unexpected, possibly critical events, such as runtime exceptions.

The logging level determines the severity level of the logged events. For example, if the logging level is set to `ERROR`, only events with the severity level `ERROR` or `FATAL` are logged.

If the logging level is set to `OFF`, logging is disabled for the given component.

Example

In the example below, debugging is enabled for the controller.

```
# Controller #
log4j.logger.org.continuent.sequoia.controller.core.Controller=DEBUG,
Console,Filetrace
log4j.additivity.org.continuent.sequoia.controller.core.Controller=false
```

9.2.2 Configure the log files to be created

By default, the following two log files are created to the `log` directory:

- `cluster.log`
- `full_cluster.log`.

You can change the logging configuration to create two additional log files:

- `request.log` - requests received by the controller
- `distributed_request.log` - execution of distributed requests (requests that must be broadcasted between the controllers).

Set the logging level to the desired value by overwriting the default value `OFF`:

- to create the `request.log` file, edit the logger definition of the `Requests` appender
- to create the `distributed_request.log` file, edit the logger definition of the `DistributedRequests` appender.

Example

In the example below, the logging level of the `Requests` appender is set to `INFO`.

```
# To trace requests #
log4j.logger.org.continuent.sequoia.controller.virtualdatabase.
request=INFO, Requests
log4j.additivity.org.continuent.sequoia.controller.virtualdatabase.reque
st=false
```

Example

In the example below, the logging level of the `DistributedRequests` appender is set to `DEBUG`.

```
# To trace distributed requests #
log4j.logger.org.continuent.sequoia.controller.distributedvirtualdatabas
e.request=DEBUG, DistributedRequests
log4j.additivity.org.continuent.sequoia.controller.distributedvirtualdat
abase.request=false
```

10 Sequoia user management

For information on the Sequoia user privileges and how they are configured in the virtual database configuration files, refer to section *Configuring Sequoia usernames and passwords* in *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

Limitations related to Sequoia user management

Existing virtual usernames (`vLogin` definitions) should not be removed from the virtual database configuration files for the following reasons:

- The Sequoia recovery log stores information on requests that have successfully executed on the backends.
- The management of the recovery log and the virtual usernames are interrelated: each entry in the recovery log is associated with a virtual username.
- Consequently, if you remove a virtual username (`vLogin` definition) from the virtual database, the automatic synchronization process when enabling a backend with the `enable` command may fail because the associated virtual username cannot be found.

Furthermore, each virtual username must be mapped to a database username (`rLogin`). If you need to remove an `rLogin`, you should create a new database dump using the `backup` command to have a consistent copy of the database that can be restored in case of problems.

Note

If an application removes an `rLogin` using the SQL query `DROP USER`, the recovery process will not be impacted because the `DROP USER` query is stored in the recovery log, and can thus be replayed along with the other entries in the recovery log.

Note, however, that if you remove an `rLogin`, virtual username definitions declared in the virtual database configuration files will no longer reflect the users in the underlying database servers.

10.1 Change a virtual database user password

Please refer to your database administration guide for the exact procedure on how to change a user password.

To change a virtual database user password

1. Shut down the first controller (`controller1`). See *3.2 Shut down a controller* on page 7 for detailed instructions.
2. Open the virtual database configuration file of `controller1` in a text editor. Edit the value of the `vPassword` attribute in the `VirtualUser` element definition to change the virtual user password.

   ```
   <VirtualUser vLogin="user1" vPassword="new_pwd"/>
   ```

3. If you also want to change the real database password, edit the corresponding `ConnectionManager` element definition.

   ```
   <ConnectionManager vLogin="user1" rLogin="realuser"
     rPassword="new_realpwd">
       <VariablePoolConnectionManager initPoolSize="20"
        minPoolSize="5"
        maxPoolSize="0" idleTimeout="180" waitTimeout="0"/>
   </ConnectionManager>
   ```

4. Change the database password on each (disabled and/or enabled) backend hosted by `controller1`.
5. Restart `controller1` and load the updated configuration file to it. See *3.1 Start a controller* on page 6 for detailed instructions.
6. Enable the backends of `controller1`.
7. Repeat steps 1-6 for `controller2`.

11 Command Line Console reference

The following sections list and describe the commands that can be executed using the Sequoia Command Line Console. Possible arguments and options that the command can receive are explained after the command syntax (see also *2.3 Command syntax notation* on page 4)

The commands are listed in alphabetical order according to the command menus of the CLC that they belong to. See section *2.2 CLC command menus and navigation commands* on page 3 for details.

11.1 Command Line Console start-up options

You can use the following options when starting the Sequoia Command Line Console with the `console.sh` script.

Option	Description
`-d` `--debug`	The CLC shows the stack trace if an error occurs.
`-e` `--exitonerror`	Stop if an error occurs in non-interactive mode.
`-f` `--file`	Specifies a file that is used as the source of commands to be executed. See also the example in *4.6 Save the virtual database configuration* on page 11.
`-h` `--help`	Displays usage information.
`-i` `--ip`	IP address of the controller host. (The default IP address is `localhost`.)
`-m` `--multiline`	Enables the use of multiline statements. By default, the use of multiline statements is disabled for backwards compatibility. If it is enabled, the statements (either single line or multiline) must be terminated by the request delimiter to be executed.
`-n,--nocolor`	Do not print colors in interactive mode for supported systems.
`-p` `--port`	JMX/RMI port number. (The default port is 1090.)

Option	Description

`-r`
`--`
`requestdelimiter`
`$string`
	Specifies the request delimiter to be used. By default, a semicolon (`;`) is used as the request delimiter.

`-s` `--secret`	Password for JMX connection.
`-u` `--username`	Username for JMX connection.
`-v` `--version`	Displays version information.

11.2 Main menu commands

The following commands can be executed from the main menu of the Sequoia Command Line Console.

11.2.1 `admin` command

The `admin` command opens `admin` mode. Note that CLC first prompts you for the administrator username and password. See sections *2.4 Command Line Console (CLC) user privileges* on page 5 and *11.3 `admin` mode commands* on page 55.

```
admin <virtual database name>
```

The `virtual database name` argument is the name of the virtual database as specified in the virtual database configuration file.

11.2.2 `load virtualdatabase configuration` command

This command loads a virtual database configuration file to the controller being managed.

```
load virtualdatabase configuration <virtual database config
file>
```

The `virtual database config file` argument specifies the file path and name of the virtual database configuration file.

11.2.3 `reload logging configuration` command

The `reload logging configuration` command loads the `log4j.properties` configuration file to the controller and updates the logging settings.

11.2.4 `save configuration` command

This command saves the current controller configuration to an XML file.

```
save configuration <filename>
```

The `filename` argument specifies the file name and path.

11.2.5 `show controller config` command

The `show controller config` command displays the current controller configuration from the `controller.xml` file.

11.2.6 `show logging config` command

The `show logging config` command displays the current logging configuration from the `log4j.properties` file.

11.2.7 `show virtualdatabases` command

The `show virtualdatabases` command lists the names of the virtual databases currently hosted by this controller.

11.2.8 `shutdown` command

The `shutdown` command shuts down the controller currently being managed. Note that the execution of this command fails if the virtual database has not been shut down using the `shutdown virtualdatabase` command.

11.2.9 `shutdown virtualdatabase` command

This command shuts down a virtual database so that it is no longer accessible on the current controller. Note that client applications can still access this virtual database on the other active controller(s). When executing this command, CLC prompts you for the administrator username and password.

Tip

You can retrieve a list of the virtual databases that are loaded into the current controller with the *`show virtualdatabases` command*. Similarly, after you shut down a virtual database, it is no longer shown when you execute the `show virtualdatabases` command.

```
shutdown virtualdatabase <virtual database name> [<mode>]
```

The `virtual database name` argument is the name of the virtual database as specified in the virtual database configuration file.

The possible values of the `mode` option are:

- `1` - before shutting down the virtual database, wait until all client connections have been closed
- `2` - before shutting down the virtual database, wait until all current transactions have been completed and all persistent connections have been closed
- `3` - do not wait until all current transactions have been completed. This option closes all connections and disables the backends without ensuring database consistency, which must be taken into account when enabling the backends.

11.2.10 `sql client` command

The `sql client` command opens the `sql client` mode, in which you can:

- issue regular SQL statements against the cluster just as with a normal RDBMS client tool
- configure certain SQL parameters, and run transactions and certain SQL scripts (see *11.6 `sql client` mode commands* on page 65 for details).

See also *2.2 CLC command menus and navigation commands* on page 3.

```
sql client <sequoia url>
```

The `sequoia url` argument is the Sequoia JDBC URL. For more information on the Sequoia URL, see section *Configuring the client application* in *Continuent.org Sequoia 2.10 Installation and Configuration Guide*.

11.3 `admin` mode commands

The following commands can be executed from the `admin` mode of the Sequoia Command Line Console.

11.3.1 `backup` command

This command creates a backup file from the database associated with a given backend using database dump. When you execute the `backup` command, CLC will prompt you for the database username and password. See *6.1 Back up a database server* on page 17.

```
backup <backend name> <dump name> <backuper>  <path>
```

The command arguments and options are:

- `backend name` - name of the backend associated with the database server that will be backed up
- `dump name` - name of the dump file to be created. Note that the `backup` command will return an error if a database dump by that name already exists.
- `backuper` - backuper that is used to create the database dump
- `path` - file path for the dump file on the controller machine

Note

For the `transfer dump` command to work correctly, you must use an identical directory structure for storing dumps in all controllers.

11.3.2 `debug` command

The debug command enables or disables the use of the debugging command set.

```
debug {on | off}
```

The command arguments are:

- `on` - enables the use of the debugging commands
- `off` - disables the use of the debugging commands.

Related topics

See also the following sections:

- *2.2 CLC command menus and navigation commands* on page 3
- *11.5 `debug` commands* on page 63

11.3.3 `enable` command

This command changes the backend state to enabled. See *5.3 Changing the backend state* on page 15.

```
enable {<backend name> | *}
```

The command arguments are:

- `backend name` - name of the enabled backend as specified in the virtual database configuration file
- `*` - enables all backend hosted by the controller that is currently being managed.

11.3.4 `expert` command

The `expert` command enables or disables the use of the expert administrator command set. See sections *2.2 CLC command menus and navigation commands* on page 3 and *11.4 `expert admin` commands* on page 60.

```
expert {on | off}
```

The command arguments are:

- `on` - enables the use of the expert administrator commands
- `off` - disables the use of the expert administrator commands.

11.3.5 `delete dump` command

This command deletes a database dump. By default, executing the `delete dump` command removes both the dump entry and the actual physical backup file from the file system.

```
delete dump <dump name> [keepfile]
```

The command arguments are:

- `dump name` - specifies the name of the database dump file to be deleted
- `keepfile` - specifies that only the dump entry is to be deleted: the actual dump file will remain on the file system.

Note

If you delete a dump file that has been shared between controllers by using the `nocopy` option of the *`transfer dump` command*, the dump file cannot be used on either controller, although the dump entry will remain visible on the associated controller. If you want to just remove the dump entry on one controller, but keep the dump file usable on the associated controller, use the `keepfile` option.

11.3.6 `disable` command

This command changes the backend state to disabled. See also *5.3 Changing the backend state* on page 15.

```
disable {<backend name> | *}
```

The command arguments are:

- `backend name` - name of the backend to be disabled as specified in the virtual database configuration file
- `*` - disables all backends hosted by the controller that is currently being managed.

11.3.7 `initialize` command

The `initialize` command is used to initialize the first backend for a new virtual database that is being activated.

```
initialize <backend name>
```

The `backend name` argument specifies the name of the initialized backend as specified in the virtual database configuration file.

Warning

The `initialize` command will completely erase the recovery log: all entries in it will be deleted. Use the `initialize` command only during initial cluster activation after Sequoia installation or to activate the cluster if the whole cluster (all controllers) have failed.

11.3.8 `purge log` command

The `purge log` command deletes the entries that were recorded in the recovery log before a specific database dump.

```
purge log <dump name>
```

The `dump name` argument specifies the point up to which the recovery log entries are deleted.

11.3.9 `restore backend` command

The `restore backend` command restores a database to a database server from a database dump file. When you execute the `restore backend` command, the CLC will prompt you for the database username and password.

```
restore backend <backend name> <dump name>
```

The command arguments are:

- `backend name` - name of the backend associated with the database server where the database dump is restored
- `dump name` - name of the dump file.

11.3.10 `show backend` command

The `show backend` command displays detailed information about the properties and current status of a given backend.

```
show backend {<backend name> | *}
```

The command arguments are:

- `backend name` -the name of the backend as specified in the virtual database configuration file
- `*` - show information about all backends of this virtual database.

11.3.11 `show backends` command

The `show backends` command lists the backends hosted by the controller that is currently being managed.

11.3.12 `show backupers` command

The `show backupers` command lists the currently available backupers that can be used to create a database backup file.

11.3.13 `show controllers` command

The `show controllers` command lists the names of the controllers hosting the current virtual database.

11.3.14 `show dumps` command

The `show dumps` command displays information about the available database dump files.

11.3.15 `show virtualdatabase config` command

The `show virtualdatabase config` command displays the virtual database configuration for the current controller.

11.3.16 `transfer dump` command

With the `transfer dump` command you can transfer a database dump from one controller to another so that it becomes available on another (remote) controller:

- by default, the data of the dump file is copied to the other controller machine
- with the `nocopy` option you can specify that the controllers share the dump through a network file system.

```
transfer dump <dump name> <controller IP>:<jmx port> [nocopy]
```

The command arguments are:

- `dump name` - name of the database dump file
- `controller IP` - IP address of the remote controller where the database dump will be copied to
- `jmx port` - JMX port number (1090 by default) of the remote controller where the database dump will be transferred to
- `nocopy` - indicates the location of the existing dump file to the other controller so that the dump file can be shared through a network file system instead of copying the data to the remote controller.

11.4 `expert admin` commands

The following commands are available from the `admin` mode of the Command Line Console after you have enabled their use with the `expert` command (see *11.3.4 `expert` command* on page 57.)

11.4.1 `clone backend config` command

The `clone backend config` command copies the configuration of a given backend in the current virtual database.

After copying the configuration of an existing backend with the `clone backend config` command, you can enable the new backend by executing the `restore backend` and `enable` commands. See also *5.4 Clone a backend* on page 16.

```
clone backend config <backend from> <backend to> <URL>
[driverPath=<value>]
```

The command arguments are:

- `backend from` - name of the backend that is cloned
- `backend to` - name of the new backend
- `URL` - database URL
- `driverPath` - the database driver class name.

11.4.2 `force checkpoint` command

You can use the `force checkpoint` command when adding to a cluster a database server node that has a consistent copy of the database. Use the `force checkpoint` command before the `enable` command to indicate to the controller which database backup is already available on this database server.

```
force checkpoint <backend name> <checkpoint name>
```

The command arguments are:

- `backend name` - name of the backend associated with the database server as specified in the virtual database configuration file
- `checkpoint name` - name of a valid checkpoint used to restore the database.

11.4.3 `force disable` command

The `force disable` command disables a backend without storing any checkpoints in the recovery log.

```
force disable {<backend name> | *}
```

The command arguments are:

- `backend name` - name of the disabled backend as specified in the virtual database configuration file
- `*` - disable all backends hosted by the controller that is currently being managed.

Warning

The database associated with the disabled backend will be in inconsistent state after a force disable operation: when you re-enable the backend, the database cannot be synchronized automatically because there is no checkpoint in the recovery log.

11.4.4 `force enable` command

The `force enable` command enables a backend without using the recovery log entries to synchronize its database with the other database server nodes in the cluster.

```
force enable {<backend name> | *}
```

The command arguments are:

- `backend name` - name of the enabled backend as specified in the virtual database configuration file
- `*` - enable all backends hosted by the controller that is currently being managed.

Warning

The database on the enabled server will be in inconsistent state after a force enable operation.

11.4.5 `force path` command

The `force path` command can be used to update the file path of a previously created database dump file.

```
force path <dump name> <file path>
```

The command arguments are:

- `dump name` - name of the dump file
- `file path` - new file path for the dump file.

11.4.6 `restore log` command

The `restore log` command copies the recovery log from the controller currently being managed to a remote controller. All previous recovery log entries on the remote controller will be deleted.

Use the `restore log` command to synchronize the virtual database recovery log on a failed controller:

- if a controller fails, no clean shutdown checkpoint can be stored to its recovery log
- because there is no shutdown checkpoint that can be used to automatically synchronize the recovery log of the failed controller, the recovery log of the remaining active controller must be copied to the failed controller using the `restore log` command.

See also *8.1 Recover from a controller node failure* on page 36.

```
restore log <dump name> <controller IP>:<jmx port>
```

The command arguments are:

- `dump name` - name of the database backup file
- `controller IP` - IP address of the remote controller where the recovery log will be copied to
- `jmx port` - JMX port number (1090 by default) of the remote controller where the recovery log will be copied to.

11.4.7 `show checkpoints` command

The `show checkpoints` command displays the checkpoints available in the recovery log. You can use this command for example to check the log id of a specific checkpoint so that you can specify the log id value in the *`dump recoverylog` command*.

11.4.8 `transfer backend` command

The `transfer backend` command transfers a backend from the current controller to a remote controller. The backend to be transferred must be in enabled state.

```
transfer backend <backend name> <controller ip>:<jmx port>
```

The command arguments are:

- `backend name` - name of the backend to be transferred as specified in the virtual database configuration file
- `controller ip` - IP address of the remote controller where the backend(s) will be transferred to
- `jmx port` - JMX port number (1090 by default) of the remote controller.

Note

The `transfer backend` command only changes the runtime configuration: the virtual database configuration files are not updated automatically. If you want to save the new configuration, use the `show virtualdatabase config` command and create new virtual database configuration files or update the existing ones.

11.4.9 `truncate log` command

The `truncate log` command deletes all entries in the recovery log before a given checkpoint.

```
truncate log <checkpoint name>
```

The `checkpoint name` argument specifies the checkpoint up to which the recovery log entries are deleted.

11.5 `debug` commands

The following debugging commands are available from the `admin` mode of the CLC after you have enabled their use with the `debug` command (see section *11.3.2 `debug` command* on page 56.)

11.5.1 `dump backend schema` command

The dump backend schema command shows the current database schema for a given backend. See also *8.3.2 Show the database schema of a database server* on page 39.

```
dump backend schema <backend name> [table name] [/columns]
[/locks]
```

The command arguments are:

- `backend name` - name of the backend as specified in the virtual database configuration file
- `table name` - restricts the output to only one table, that is, use this option to display the columns and locks in a given table
- `/columns` - outputs all column names
- `/locks` - outputs the transaction ID(s) that are currently locking the table(s).

11.5.2 `dump parsing cache` command

The `dump parsing cache` command shows the configuration of the parsing cache and lists the entries in it, including the requests that have been / are currently being parsed. The contents of the parsing cache are displayed one screen at a time, allowing you to scroll the output.

See also *8.3.5 Show the request parsing cache configuration and entries* on page 43.

11.5.3 `dump queues` command

The `dump queues` command shows the pending requests for a specific backend. See also *8.3.4 Show the pending requests for a backend* on page 42.

```
dump queues <backend name>
```

The `backend name` argument is the name of the backend as specified in the virtual database configuration file.

11.5.4 `dump recoverylog` command

The `dump recoverylog` command shows the contents of the recovery log. The contents of the recovery log are displayed one screen at a time, allowing you to scroll the output.

```
dump recoverylog {[indexes] | [<min>] [<max>]}
```

The command arguments are:

- `indexes` - shows the minimum and maximum indexes and the number of entries in the recovery log
- `min` - the minimum index (checkpoint ID) starting from which the contents of the recovery log are shown
- `max` - the maximum index (checkpoint ID) up to which the contents of the recovery log are shown.

See also *8.3.1 Show the contents of the recovery log* on page 38 and *11.4.7 `show checkpoints` command* on page 62.

11.5.5 `dump request` command

The dump request command shows detailed information about a specific request currently pending in the request scheduler queue.

```
dump request <request id>
```

The `request id` argument specifies the ID of the request: you can use the *`dump scheduler queues` command* to show a list of all pending requests and their IDs. See also *8.3.3 Show information about pending requests and transactions* on page 40.

11.5.6 `dump scheduler queues` command

The `dump scheduler queues` command shows a list of currently pending:

- transactions
- read requests
- write requests.

See also *8.3.3 Show information about pending requests and transactions* on page 40.

11.5.7 `parse request` command

The parse request command calls the Sequoia request parser with a given SQL request and show the expected result of this request parsing. The parsing lists the modifications that would occur on the database if the request was executed.

See also *8.3.6 Show the request parsing results* on page 45.

11.6 `sql client` mode commands

The `sql client` mode allows you to interact with the cluster using regular SQL statements, just as with a normal RDBMS client tool.

You can also use the `sql client` mode to configure certain SQL parameters, and to run transactions and certain SQL scripts. See the following command descriptions for more details.

Note

Command line input other than the commands listed below is interpreted as an SQL statement.

11.6.1 `begin` command

The `begin` command starts a transaction.

11.6.2 `commit` command

The `commit` command commits a transaction.

11.6.3 `fetchsize` command

The `fetchsize` command sets the fetch size value for future statements.

```
fetchsize <number of rows>
```

11.6.4 `load` command

The `load` command can be used to execute SQL statements from a file.

```
load <file name>
```

11.6.5 `maxrows` command

The `maxrows` command can be used to specify the maximum number of rows to be returned from the database.

```
maxrows <number of rows>
```

11.6.6 `rollback` command

The `rollback` command rolls back a transaction to a given savepoint.

```
rollback <savepoint name>
```

11.6.7 `savepoint` command

The `savepoint` command creates a named transaction savepoint.

```
savepoint <savepoint name>
```

11.6.8 `setisolation` command

The `setisolation` command can be used to set the connection transaction isolation level.

```
setisolation <isolation level>
```

The isolation level can be:

- `0 - TRANSACTION_NONE`
- `1 - TRANSACTION_READ_UNCOMMITTED`
- `2 - TRANSACTION_READ_COMMITTED`
- `4 - TRANSACTION_REPEATABLE_READ`
- `8 - TRANSACTION_SERIALIZABLE`

11.6.9 `showtables` command

The `showtables` command displays all tables in the current virtual database.

11.6.10 `timeout` command

The `timeout` command can be used to set the query timeout value. The default value is 60 seconds.

```
timeout <seconds>
```