

# CUDB Data Storage Handling

---

## FACILITY DESCRIPTION

**Copyright**

© Ericsson AB 2016, 2017. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

**Disclaimer**

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

**Trademark List**

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scope	1
1.2	Revision Information	1
1.3	Target Groups	2
1.4	Prerequisites	2
1.5	Typographic Conventions	2
<b>2</b>	<b>Overview</b>	<b>3</b>
2.1	Prerequisites	3
2.2	Architecture	3
2.3	Description	5
2.4	Dependencies and Interactions	23
<b>3</b>	<b>Operation and Maintenance</b>	<b>29</b>
3.1	Configuration	29
3.2	Fault Management	34
3.3	Performance Management	39
3.4	Security	40
3.5	Logging	41
	<b>Glossary</b>	<b>45</b>
	<b>Reference List</b>	<b>47</b>





# 1 Introduction

This document provides a description for the data storage handling features of Ericsson Centralized User Database (CUDB).

## 1.1 Scope

The purpose of this document is to describe the architecture, feature interactions, and main functions of the data storage handling features, including the various data backup, restore, and reconciliation procedures. In addition, the document also provides information on the operation and maintenance of the feature.

## 1.2 Revision Information

This section contains the changes in the feature between the releases of this document.

### **Rev. A**

This document is based on 8/155 34-HDA 104 03/9 with the following changes:

- Updates throughout the document on configurations containing nodes without a local Processing Layer Database.
- Updated terminology throughout the document, including figures.

### **Rev. B**

Other than editorial changes, this document has been revised as follows:

- Updated Ericsson personnel information.

### **Rev. C**

Other than editorial changes, this document has been revised as follows:

- Section 3.4 on page 40: Updated the security measures with Access Control.

### **Rev. D**

Other than editorial changes, this document has been revised as follows:



- Section 3.4 on page 40: Updated Access Control security measures.

#### **Rev. E**

Other than editorial changes, this document has been revised as follows:

- Updated Ericsson personnel information.

## 1.3 Target Groups

This document is intended for system administrators and users working with the data storage handling features, and performing data backups, restore operations, or data reconciliation.

## 1.4 Prerequisites

Users of this document must have basic knowledge and experience of the following:

- Relational DBMS concepts.
- Distributed Computing concepts.
- Networked File Systems concepts.

## 1.5 Typographic Conventions

Typographic conventions can be found in the following document:

- *Typographic Conventions*.



## 2 Overview

CUDB is a distributed, highly resilient, in-memory database, providing a seamless, geographically-distributed database system intended as a common repository in the network to store subscriber data for diverse applications.

As a physically-distributed system, CUDB is made up of a number of nodes. The CUDB node is the main architectural unit of a CUDB system. Each CUDB node is supposed to be deployed in one single network location or site for redundancy reasons.

All CUDB nodes are interconnected through a reliable IP transport network. Data stored in the CUDB system is exposed as a Lightweight Directory Access Protocol (LDAP) Directory Information Tree (DIT). Refer to *CUDB LDAP Interwork Description*, Reference [1] for more information on data structures within the LDAP tree.

### 2.1 Prerequisites

This section is not applicable to this feature.

### 2.2 Architecture

From a logical point of view, the architecture of a CUDB system is made up of two data layers, as shown in Figure 1.

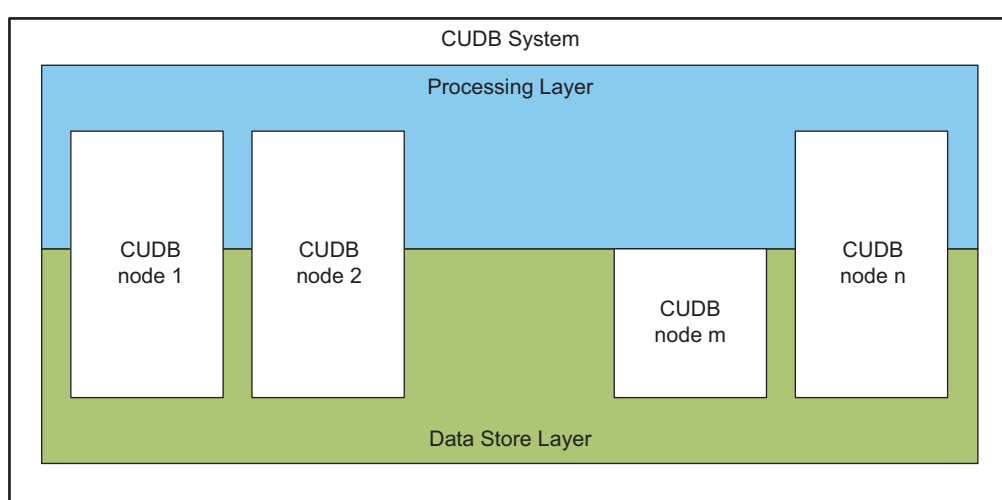


Figure 1 CUDB Architecture, Featuring Two Data Layers



The Data Store (DS) Layer spans all the CUDB nodes in the system, while the Processing Layer (PL) not necessarily spans all of them. Components of each layer can communicate with other components of the same layer, even if they are located in different network locations. Exceptions are the CUDB nodes without PL, where they communicate with one PL in the same site. The PL elements can only access DS Layer components located in the same node. For example, if a PL element in CUDB node A needs some resources of a component of the DS Layer in CUDB node B, it must communicate with the PL elements in CUDB node B to access the DS Layer component from there to get the data. See the subsections below for more information on the two layers.

### 2.2.1 Data Store Layer

The DS Layer is the main data repository in CUDB. The entire subscriber dataset stored in CUDB is partitioned into several Data Store Unit Groups (DSGs), each storing a disjoint partition of subscriber data (see Figure 3).

For each of those data partitions, CUDB allows one, two or three instances (or replicas) to be configured, as described in *CUDB Technical Product Description*, Reference [2]. Each replica of a DSG is stored in a different CUDB node and typically in a different site. The number of DSGs in a CUDB system, the number of replicas, and the distribution of replicas across the CUDB system depend on the system configuration. Refer to *CUDB Data Distribution*, Reference [3] for more information.

### 2.2.2 Processing Layer

In some configurations, the PL is not present on all the CUDB nodes. The PL supports the LDAP protocol handling, and has the application logic to resolve any request. For this purpose, the PL on each node includes a pool of LDAP Front Ends (FEs), and an instance of the Processing Layer Database (PLDB).

Refer to *CUDB Data Distribution*, Reference [3] for more information on PLDB.

### 2.2.3 Geographical Redundancy and Replication

For each of the data partitions—PLDB and the DSGs—one of the replicas is considered to be the authoritative source of data. Such replicas are called **master replicas** and all other replicas of that data partition, if any, are called **slave replicas**. All write operations will be addressed to master replicas, then replicated asynchronously to slave replicas. If the master replica of the PLDB or of any DSG fails or becomes unreachable for any reason, another replica of the corresponding data partition is appointed as the new master replica, if any is available. Refer to *CUDB High Availability*, Reference [4] for more information.

For redundancy purposes, two replication channels connect the master replicas to the slave replicas. These channels work in active-standby redundancy mode.





For more information on replication mechanisms and the handling of master and slave replicas, refer to *CUDB High Availability*, Reference [4].

## 2.3 Description

This section contains detailed information about data storage handling in CUDB.

### 2.3.1 Data Storage

CUDB Data Storage is realized primarily through the Storage Components acting as a repository and database in CUDB. Storage Component units are in-memory cluster database systems made up of a set of blades or Virtual Machines (VMs) behaving as a unit, and include all the PLDB and DSG databases of the CUDB system. For a simplified physical and logical view of the CUDB node, see Figure 2.

CUDB supports defining Binary Large Objects (BLOB) in the data model. It is possible to select whether each type of BLOB is stored in the memory or the disk storage system. Refer to *CUDB Binary Large Object Attributes Management*, Reference [5] for more information.

A PLDB is a variable-size cluster running on 2 to 16 database blades or VMs. Database clusters storing DSG data are called DS Units. A DS unit is a fixed-size cluster, running on two database blades or VMs.

The number of DS units in a CUDB node is determined by the total number of blades or VMs in the node and the number of blades or VMs used by PLDB (in some configurations the latter is 0 in nodes without PLDB). Therefore, a CUDB node may host as many DS units as the number of remaining pairs of payload blades or VMs, that is, excluding the two System Controllers (SCs).

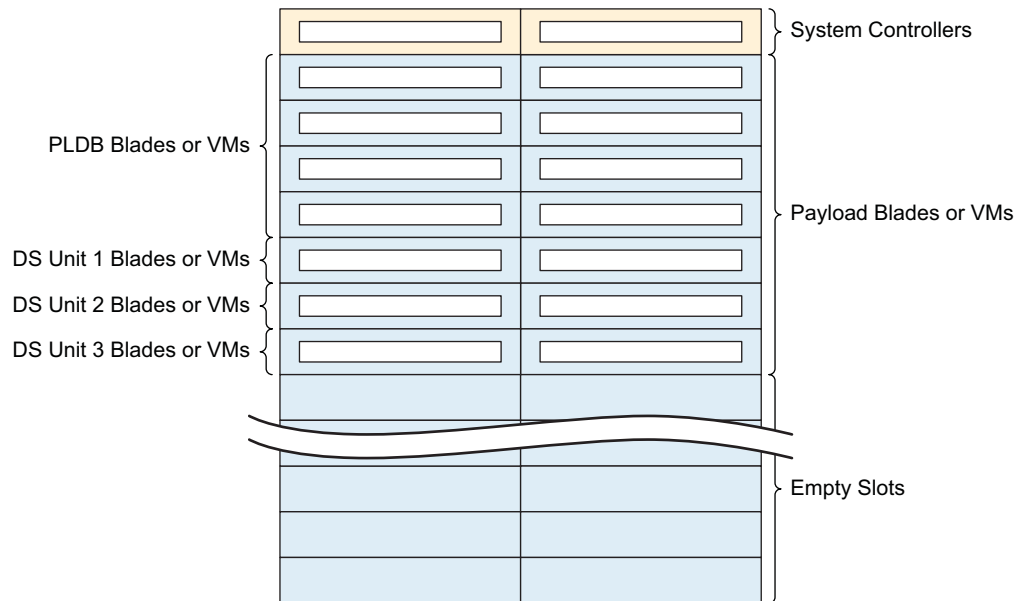


Figure 2 Simplified Physical and Logical View of a CUDB Node

The CUDB Storage Component is characterized by the following attributes:

- **Redundancy**

The CUDB Storage Component provides data redundancy mechanisms on two levels:

- Local redundancy of the data is ensured by configuring database clusters storing PLDB and DSG data to hold two copies of every piece of data they store. The underlying database technology keeps these two intra-cluster replicas consistent by means of a synchronous replication mechanism.
- Geographical redundancy is a feature which ensures that the data is stored in the entire CUDB system by using the above-mentioned distribution and replication mechanisms. In this case, asynchronous replication is used between peer CUDB storage components. Refer to *CUDB High Availability*, Reference [4] for more information on geographical redundancy.

- **Persistency**

Persistency is achieved through storing data in the memory of two servers of any clustered system (DS Unit or PLDB). In addition, data also persists on the local disk storage system in each server by flushing them every few milliseconds. Persistency is also guaranteed by the backup and restore mechanism provided by CUDB (see Section 2.3.7 on page 12 for more information).

- **Consistency**



Intra-cluster and replica-to-replica consistency is achieved as follows:

- The local redundancy mechanism ensures consistency by not registering a transaction until the change has been implemented in the peer database nodes within a database unit (PLDB or DS Unit). If this cannot be done, the transaction is aborted and rolled back. However, if the process succeeds, the data is stored twice in the database cluster.
  - The replication mechanism ensures consistency between replicas. A proprietary replication mechanism ensures consistency by means of master/slave based asynchronous mechanism.
- **Load sharing**

Each DS Unit, that has two data cluster members, has its data set distributed in a way that half of it is handled primarily by one of those cluster data members. This allows the load sharing of the data store requests amongst the two members of the cluster. At the same time, although PLDBs can have more than two data cluster members, the load distribution is the same for every peer of data cluster member.

## 2.3.2 Processing Layer Database Distribution

The PLDB can be configured to be hosted in all the nodes of the system or only in some of them, but at least one replica of the PLDB must be configured in a node for every site in the system. The advantage of not configuring the PLDB in all the nodes of the site is to free up some blades or VMs for extra DSGs, so this scenario is mainly used for large deployments. Refer to *CUDB Node Configuration Data Model Description*, Reference [6] for more information on PLDB configuration.

## 2.3.3 Data Storage Distribution

A DS Unit can be configured to host data for any single DSG. As shown in Figure 3, DS Unit 2 hosts data for DSG 2 in CUDB Node 1, but the same DSG 2 is hosted by DS Unit 1 in CUDB Node 5. DSGs do not necessarily need to be hosted in all CUDB nodes: the data stored in all the DSGs is accessible from any CUDB node in the system.

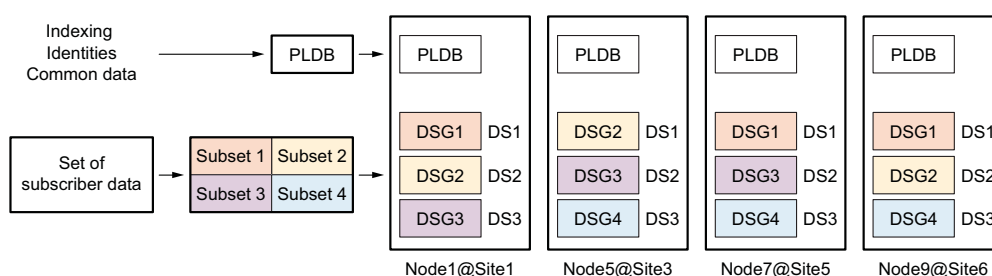


Figure 3 Data Distribution Example



As shown in Figure 4, nodes without PLDB can host more DS units than CUDB nodes with PLDB: the blades or VMs hosting the PLDB are converted to host DS units. The currently valid DS unit distribution rules for the CUDB nodes must be followed for these new DS units.

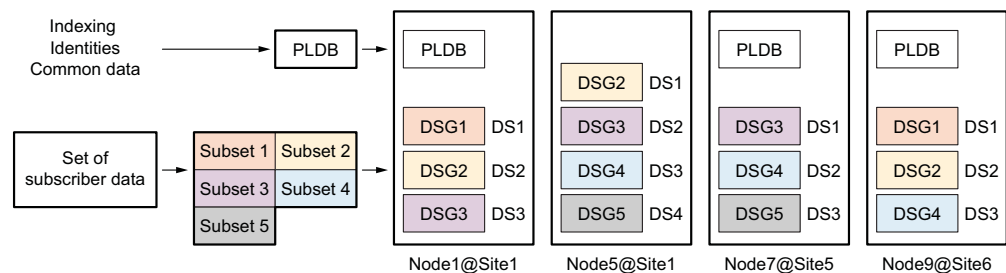


Figure 4 Data Distribution Example with a Node without PLDB

Internally, each data partition is organized in a relational model as provided by the Storage Component. Externally, the data storage is accessed with the LDAP protocol as a Directory Information Tree (DIT). This internal relational model assigns memory chunks to individual tables indexed by object classes.

The overall memory occupation levels in the PLDB and DS Units are monitored. When the occupation levels reach a certain threshold, an alarm is raised.

These alarms are raised if the warning level (configured with the `memoryWarningThreshold` parameter) and the overall maximum level (configured with the `memoryFullThreshold` parameter) are reached. It is strongly recommended not to change the full threshold value. The warning threshold values can be freely configured to satisfy safety requirements. See Section 3.2 on page 34 for more details.

In addition, batches of subscriber data can be moved from one DSG to a different DSG to provide a more balanced memory occupation or a more balanced load distribution across DSGs. This process is known as the subscription reallocation, or “reallocation” procedure. Refer to *CUDB Subscription Reallocation*, Reference [7] for more information.

Data can be distributed among DSGs in accordance to the geographical zones as well. Refer to *CUDB Multiple Geographical Areas*, Reference [8] for more information.

### 2.3.4 Data Storage Management

The configuration model of a CUDB node includes information about PLDB, all replicas of PLDB in the CUDB system, all DSGs and all their replicas (DS Units) in the CUDB system. Refer to *CUDB Node Configuration Data Model Description*, Reference [6] for more details on this configuration model. Data store management is carried out both through changes in the configuration data model and by executing CLI commands.



When managing data partitions, the following procedures must be considered:

- DS Unit Group creation
- DS Unit Group activation
- DS Unit addition
- DS Unit activation
- DS Unit deactivation
- PLDB Unit activation
- PLDB Unit deactivation
- Set PLDB in maintenance mode
- Set a DS Unit in maintenance mode

See Section 3 on page 29 for details about these management tasks.

### 2.3.5 Data Storage Scalability

The number of database blades or VMs to use for PLDB determines its capacity and cannot be changed. It is crucial to properly dimension PLDB during initial installation to accommodate the actual and future required capacity.

Storage space for subscriber data and processing capacity to handle that extra data can be increased by adding new DSGs, along with the corresponding DS Units, to the CUDB system.

A CUDB system can accommodate up to 255 DSGs.

### 2.3.6 Reconciliation

Reconciliation is a system-wide process whose goal is to detect and correct discrepancies among the PLDB and DSGs. Since CUDB is an application-agnostic system, the discrepancies that the reconciliation process is able to detect and correct are simply entries stored in a DSG that have no matching Distribution Entry (DE) in the PLDB as an ancestor. Such a discrepancy is known as **inconsistency** in CUDB. Reconciliation fixes inconsistencies by deleting the offending data from the DSG where it is hosted.

**Note:** Reconciliation runs as a low priority background task to minimize its impact on traffic processing. It is executed in nodes where there are DSG master replicas which are suspected to be inconsistent with PLDB. If there are several DSG master replicas to be reconciled in a CUDB node, they are handled one by one.



### 2.3.6.1 Reconciliation Tasks

The reconciliation process runs on every CUDB node, and executes the following four main tasks:

- Reacts to mastership elections in the CUDB system.
- Manages the list of local DSG replicas marked for reconciliation by automatically adding and removing DSGs to the list stored in the local PLDB for nodes with PLDB. In nodes without local PLDB, the reconciliation process adds and removes tasks to the list allocated in the master PLDB replica.
- Sequentially reconciles every DSG in the list with PLDB until all of them have been reconciled. Reconciliation takes place in the CUDB node where the master replica of the reconciled DSG is. A reconciliation task for a given DSG ends in one of the following outcomes:
  - When all entries are checked and no discrepancies are found.
  - When all discrepancies in that DSG have been located and corrected.
  - When the replica of that DSG (DS Unit) where reconciliation is being run ceases to be the master replica for its DSG halfway through reconciliation. In this case, the new master replica for that DSG is marked for reconciliation.
- Raises and clears alarms related to data reconciliation:

A `Storage Engine, Temporary Data Inconsistency` alarm is raised for each DSG master replica marked for reconciliation in the CUDB node where that master replica is. Once reconciliation in a DSG master replica is finished, the alarm is automatically cleared. See Section 3.2.4 on page 36 for more information.

If data from a DSG has been deleted due to reconciliation, the `Storage Engine, Deleted Data due to Reconciliation` alarm is raised for that DSG in the CUDB node where that master replica is. This type of alarm must be manually cleared. See Section 3.2.4 on page 36 for more information.

### 2.3.6.2 Reconciliation Scenarios

Due to asynchronous replication and the possibility to do a group data restore, discrepancies among the PLDB and the DSGs can appear in a CUDB system. These discrepancies are usually due to the side effects of the layered data distribution of CUDB (PLDB in the PL, and the DSGs in the DS Layer), the geographical redundancy of the stored data, and the asynchronous nature of replication in the underlying database engine.

There are five different scenarios where reconciliation is needed which are as follows:



- **Reconciliation after PLDB mastership election**

A mastership change in the PLDB can affect every DSG in the system. Therefore, in case of PLDB mastership election, the master replicas for every DSG are marked for reconciliation at the CUDB node where they are hosted. Added DEs might be missing from the newly elected PLDB master replica.

- **Reconciliation after DSG mastership election**

A DSG mastership change affects only the data stored in that individual DSG. Therefore, in case of DSG mastership election, only the master replica of the affected DSG is marked for reconciliation at the node where it is hosted. Deleted subtrees below DEs might still be present in the newly elected DSG master replica.

- **Reconciliation after group data restore for PLDB**

After a group data restore for PLDB, reconciliation works if a new master replica has been elected for PLDB. The restored PLDB data is older than the data in DSGs and will not include DEs that were created after the backup used for the restore operation.

- **Reconciliation after group data restore for DSG**

After a group data restore for a DSG, reconciliation works as if a new master replica has been elected for that DSG. The restored DSG data is older than the PLDB data and will still have subtrees below DEs that were deleted after the backup used for the restore operation.

- **Reconciliation after CUDB system data restore**

After the successful restoration of the CUDB system, a mastership election takes place for PLDB and for every DSG, therefore the master replicas of all DSGs are marked for reconciliation.

Reconciliation can also be executed manually by the `cudbReconciliationMgr` command. Refer to *CUDB Node Commands and Parameters*, Reference [9] for more information.

**Note:** In case of group data restore and system data restore operations, reconciliation must be scheduled manually, unless the automatic execution of Selective Replica Check and Data Repair processes is disabled by configuration. Check the details of the `CudbDsGroupRepairAndResync` class in *CUDB Node Configuration Data Model Description*, Reference [6] for more information about configuring these processes.



### 2.3.6.3 Reconciliation Output Files

If the reconciliation process finds inconsistencies between the PLDB and a specific DSG, a file containing the deleted data is generated for logging purposes.

**Note:** These output files are meant for Ericsson support personnel only.

### 2.3.6.4 Reconciliation Management

Reconciliation is automatically handled by the CUDB system and requires no administrative actions other than managing raised alarms.

The `cudbReconciliationMgr` command is available to interact with the reconciliation feature, by offering the following functions:

- Printing the list of DSGs marked for reconciliation.
- Checking if a specific DSG is marked for reconciliation.
- Marking a specific DSG for reconciliation.

**Note:** DSGs can only be marked for reconciliation if their master replicas are hosted locally.

Refer to *CUDB Node Commands and Parameters*, Reference [9] for more information on this command.

## 2.3.7 Data Backup and Restore

Data Backup dumps the contents of the databases into files, which can later be used to recover the contents of the databases when needed. Data backups can be created both of a particular database partition (unit data backup) and of the complete CUDB system (system data backup).

A CUDB system data backup (that is, a backup of all the data in the CUDB system) is composed of single backups of every data partition inside the CUDB, that is, PLDB and all DSGs. Unit data backups are useful in specific recovery situations.

It is recommended to create a system data backup whenever any major system change takes place.

The restore types are similar to the backup types: CUDB offers the restoration of the complete CUDB system (system data restore), of all replicas of a data partition (group data restore), and of a particular database unit (unit data restore).

The backup and restore functions are described below in more detail.





### 2.3.7.1

#### Unit Data Backup

Backing up a single database unit is known as the unit backup, and it must be ordered in the node that holds the master PLDB or DSG replica of the unit.

**Note:** Before initiating a backup, operators must locate the master replica of the unit by using the `cudbSystemStatus` command. Refer to *CUDB System Administrator Guide*, Reference [10] for more information.

Once the node where the backup must be initiated is chosen, the backup is executed using the `cudbManageStore` command.

The output of the unit backup consists of three files for each participating data node in the cluster (two data nodes in case of DSGs, 2 to 16 data nodes in case of PLDB, unless any of the data nodes in the cluster is down - refer to *CUDB High Availability*, Reference [4] for more information). These files are as follows:

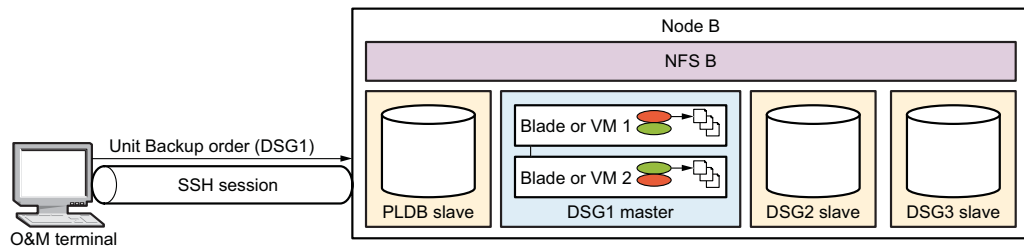
- Metadata file (`.ctl` extension).
- Table records file (`.data` extension). This file is compressed.
- Transaction log file (`.log` extension).

The files generated by the unit backups are stored within the local disk storage system of each participating blade or VM. The files are located in the following directory:

`/local/cudb/mysql/ndbd/backup/BACKUP/BACKUP-YYYY-MM-DD_HH-mm`, where YYYY, MM, DD, HH, and mm stand for the year, month, day, hour, and minute of backup execution, respectively.

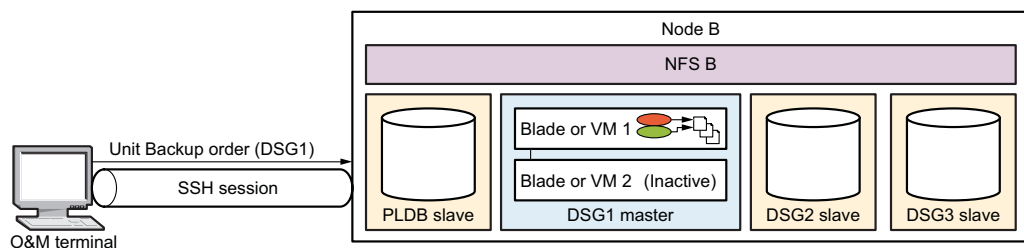
**Note:** To conserve space in the local disk storage system, the blades or VMs keep only the files of the last three unit backups. Subsequent unit backups ordered on the same unit result in deleting the oldest unit backup files to make room for the new backup files.

Figure 5 shows the inside of the DSG1 master replica unit (part of Node 1 shown in Figure 3), and how both blades or VMs hold the complete data set in the unit. If both blades or VMs are active, each one dumps one part of the complete data set into the backup files, so the overhead caused by the backup operation is evenly spread among both. The colored ovals represent the sub-sets into which the data set (held by the unit) was split.



**Figure 5** Inside View of DSG1 during Database Unit Backup (DSG1 Serves as Master Replica, All Blades or VMs Active)

However, it can happen that when the backup is ordered, one of the blades or VMs running the unit marked for backup is down. In that case, that blade or VM does not generate any files at all, but dumps the data of the inactive blade or VM into the backup files that are generated (because its peer contains the whole data set in the replica). Figure 6 shows what happens in such a case.



**Figure 6** Inside View of DSG1 during Database Unit Backup (DSG1 Serves as Master Replica, One Blade or VM Inactive)

In case of problems during the backup procedure, the alarm corresponding to the PLDB and DSG backup is raised. See Section 3.2.2 on page 35 for more information.

### 2.3.7.2

#### System Data Backup

Backing up the data of the complete CUDB system can be ordered on any CUDB node with the `cudbDataBackup` command. A system data backup is made of a unit data backup of the PLDB and a unit data backup of each of the DSGs in the system. The clusters where the data unit backups are taken are chosen according to the following rules:

- The backup of PLDB is performed always in the master replica.
- Backups of DSGs are performed in a slave replica if any is available (configured and non-degraded); if no slave replica is available for a given DSG, the master replica is used instead. The replicas to use for the system data backup are selected in a way that tries to avoid taking all backups of all DSGs in the same CUDB node.



System data backup cannot be initiated in the following cases:

- No master PLDB replica exists.
- The master PLDB replica is degraded (refer to *CUDB High Availability*, Reference [4] for more information on degradation).
- All slave replicas and also the master replica of any DSGs are down or degraded in a geographically redundant system.

Since CUDB is a highly-distributed database system, and (as mentioned above) the system-level backup consists of a series of individual data backups for the PLDB and every DSG database, the output of this backup is a number of individual backups spread over the participating CUDB nodes of the system. However, the individual unit backups of the system-level backup are not interchangeable with the single database unit backups described in Section 2.3.7.1 on page 12. The unit backups of the system data backup procedure differ from the single database backups in the following:

- In case of a system-level data backup, the output files of the unit data backups (.ctl, .data, and .log) are compressed into a TAR file. One TAR file is generated for each participating blade or VM.
- The output files of the unit data backups are generated in the local disk storage system of every participating blade or VM, and the TAR files are stored in a Network File System (NFS) directory shared by all the blades or VMs in the CUDB node. This way, administrators can then easily access those TAR files at a common place, and move them to another node or to a safer storage.
- The output files of the unit data backups are removed from the local disk storage system, once they are compressed into the TAR file. The exceptions are the system data backup interruption or failure, when the temporary output files of the unit data backups are not deleted. To prevent undesired unit data backup pieces spreading over the local disk storage system, the system-level data backup procedure can clean all the remaining (previously-generated) backup pieces of individual backups before starting with the new individual backups.

**Note:** This procedure does not affect the files of single database unit backups.

Every TAR file generated for each participating blade or VM is placed in the `/home/cudb/systemDataBackup` directory within the NFS storage. The format of the TAR filename is as follows:

`BACKUP-<DateTime>-<CUDBNodeId>.<Unit>.<ClusterNodeId>.tar`

The individual filename elements in the backup filename stand for the following:

<b>DateTime</b>	Contains the date and time when the backup was executed. It follows the YYYY-MM-DD_HH-mm format, where YYYY, MM, DD, HH, and mm stand for the year, month, day, hour, and minutes, respectively.
-----------------	--



<b>CUDBNodeId</b>	The CUDB Node Identifier of the node where the backup was executed.
<b>Unit</b>	The unit number of the PLDB or the DSG where the DS belongs. Its value is 0 for PLDB and 1-255 in case of DSGs.
<b>ClusterNodeId</b>	The database cluster node ID, used to identify the data node within the cluster.

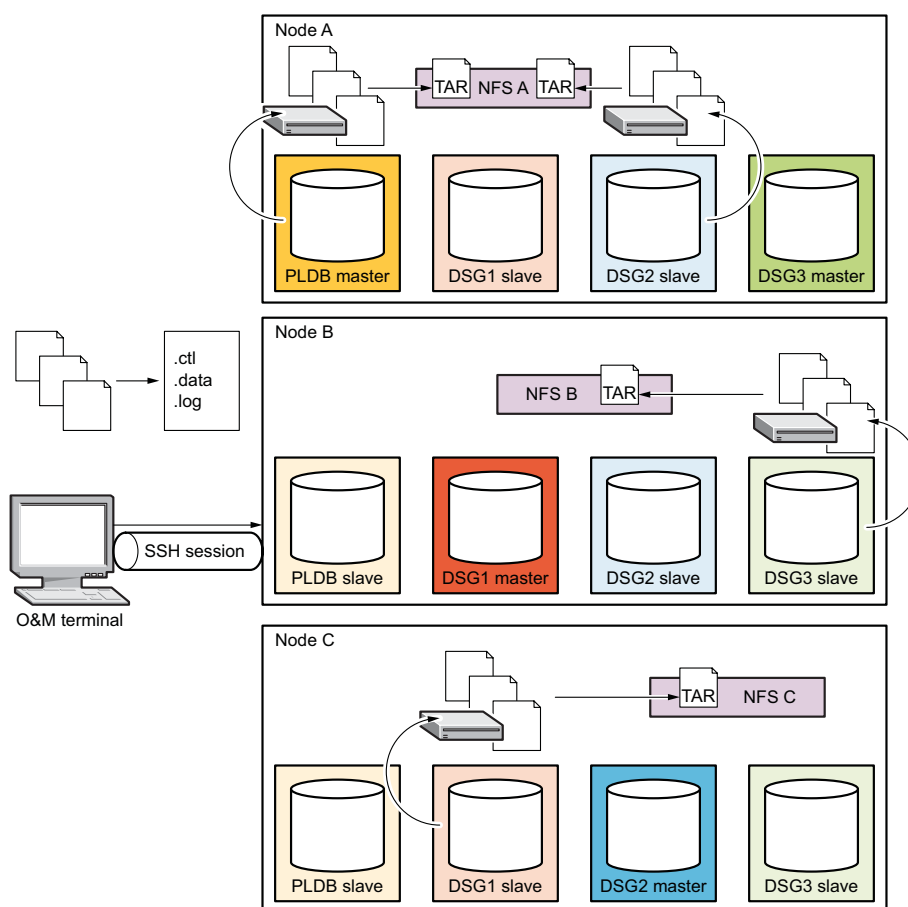
**Note:** To conserve space in the disk storage system, the system keeps only the files of the last three system backups in the shared directory of the CUDB node. Subsequent system backups result in deleting the oldest system backup to make room for the new backup files.

During the execution of the system data backup, the information about the chosen DSG database replicas is printed out to the standard output of the SSH session through which the system data backup command was entered. It is recommended to keep this information, since administrators need it to collect the backup files from the different CUDB nodes before initiating a system restore. See Section 2.3.7.7 on page 19) for more details.

As explained in Section 2.3.7.7 on page 19, system data restore results in the loss of all changes in the CUDB database occurred between the creation of the system data backup and the execution of the system data restore. Therefore, administrators must be completely sure that the restoration is really necessary.

Individual PLDB and DSG database backups in a system data backup are not synchronized. This means that data in those individual backups might not be fully aligned if provisioning is on while the system data backup is being created (for example, in case new subscriber information is added to the PLDB and the DSGs). Therefore, operators may choose to stop provisioning during system data backup. For more information, refer to *CUDB Backup and Restore Procedures*, Reference [11].

Figure 7 shows how a complete system data backup is executed in the entire CUDB system, and what the outcome of the backup execution is.



**Figure 7** Complete System Data Backup in a CUDB System. System Contains 3 DSGs, Deployed in Geographical Redundancy

System data backups have some interactions that must be considered in order to minimize inconsistencies in the content of backup files. See Section 2.4 on page 23 for further details.

In case of problems during the backup procedure, the alarm corresponding to the PLDB and DSG backup is raised. See Section 3.2.2 on page 35 for more information.

### 2.3.7.3

#### Periodic System Data Backup

System data backups can be automated by creating periodical system data backup tasks, using the Linux Distribution Extension (LDE) `cron` resource. This way, administrators can ensure that a recent backup is always available without the need of manual intervention. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information.



The alarms that apply to the periodic backup are the same as the normal backup of a complete CUDB system. See Section 3.2.2 on page 35 for more information.

#### 2.3.7.4 Unit Data Restore

The restore of a single database unit is called the unit data restore, and makes it possible to recover a unit data backup into a single database replica (see Section 2.3.7.1 on page 12 for more information on unit data backups). The operation is executed with the `cudbManageStore` command.

**Note:** It is strongly recommended to use only the latest unit data backup for restore operations. Using obsolete unit data backups for unit restore operations can lead to the impossible establishment of the replication that can only be fixed with making a new unit data backup and restoring that again. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information.

Before executing the restore operation, the output files of the latest unit data backup must be copied to the destination blades or VMs of the replica. As described in Section 2.3.7.1 on page 12, the backup files are located by default in the following directory of the local disk storage system of the blade or VM:

```
/local/cudb/mysql/ndbd/backup/BACKUP/BACKUP-YYYY-MM-DD_HH-mm
```

Then the files must be copied to the blades or VMs where the cluster data nodes of the unit to restore are running. In case not all the blades or VMs of a unit are available, refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information.

After the successful execution of the restore operation, the replication is started automatically. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for the exact steps of performing unit data restore.

**Note:** Stored SQL procedures issued in the database for different purposes (such as performance management) are neither backed up, nor restored. Therefore, stored SQL procedures intended to run in the CUDB system must be recovered individually at every unit. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more details.

In case of problems during the restore procedure, the corresponding DS and PLDB alarms are raised. See Section 3.2.3 on page 36 for more details.

#### 2.3.7.5 PLDB Group Data Restore

In case the restoration of all PLDB replicas is required at once, the state of the PLDB data can differ from the state of the DSG data after successful restoration, the latter being more recent.



In such case, when reconciliation is executed either automatically or manually, it might remove the recently added subscriber data, thereby restoring all the DSG replicas back to the state when the PLDB was backed up. See Section 2.3.6 on page 9 for more details on reconciliation.

**Note:** If the Selective Replica Check and Data Repair processes are enabled, then reconciliation must be scheduled manually for all DSGs after a PLDB group data restore. Check the details of the `CudbDsGroupRepairAndResync` class in *CUDB Node Configuration Data Model Description*, Reference [6] for more information about configuring the Selective Replica Check and Data Repair processes.

### 2.3.7.6 DSG Group Data Restore

The restore of a complete DSG database is also known as the group data restore. It consists of the unit data restore of all the DSG database replicas.

Replication must be started manually on all DSG slave replicas. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more details on the process, and the exact steps to follow.

**Note:** Group data restore initiated for a DSG does not affect the rest of the DSGs in the CUDB system. However, following the master election of the restored group, a reconciliation procedure is executed automatically, which removes the inconsistent entries of the restored group from the system (since the restored data is likely outdated compared to the current PLDB). See Section 2.3.6 on page 9 for more information about reconciliation.

Similarly to the unit data restore procedure, the stored SQL procedures must be recovered individually at every unit. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more details.

In case of problems during the restore procedure, the corresponding DSG alarm is raised. See Section 3.2.3 on page 36 for more details.

**Note:** If the Selective Replica Check and Data Repair processes are enabled, then reconciliation must be scheduled manually for the restored DSG after a DSG group data restore. Check the details of the `CudbDsGroupRepairAndResync` class in *CUDB Node Configuration Data Model Description*, Reference [6] for more information about configuring the Selective Replica Check and Data Repair processes.

### 2.3.7.7 System Data Restore

The restore of the entire CUDB system is also known as the CUDB system data restore. This type of restore requires a previous CUDB system data backup, and is equivalent to the individual restoration of all database system units (that is, the restoration of all PLDB replicas and all DSGs database replicas). System data restore can only be executed on a system that is exactly the same as the system from which the backup was taken.



System data restore results in the loss of all changes in the CUDB database occurred between the creation of the system data backup and the execution of the system data restore. Therefore, administrators must be completely sure that the restoration is really necessary.

**Note:** Restoring a complete CUDB system wipes out all the changes in the system since the last system backup.

The detailed steps of performing a complete system restore are available in *CUDB Backup and Restore Procedures*, Reference [11]. The following list provides an overview of the steps to perform during the process.

1. File gathering.

The files to use for the system restore operation must be copied from their original location (see Section 2.3.7.2 on page 14) to the NFS storage of each CUDB node.

2. File distribution.

The files to restore must be copied and extracted from the NFS storage of the nodes to the persistent disk storage system of every blade or VM.

3. Executing restore operation.

Once all files are located on the proper blades or VMs, the restore can be executed with the `cudbSystemDataBackupAndRestore` command.

4. Executing replication.

After the restore is finished, replication must be started manually. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more details.

5. Recovering SQL procedures.

Similarly to the unit data restore and group data restore, the stored SQL procedures must be recovered individually at every unit. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more details.

In case of problems during the restore procedure, the corresponding restore fault in PLDB and DS alarms are raised. See Section 3.2.3 on page 36 for more details.

### 2.3.7.8

#### Combined Unit Data Backup and Restore

The automated procedure for a single database unit backup and restore enables to execute these two steps with a single command, using the `cudbUnitDataBackupAndRestore` script. The command takes a unit data backup on the master replica and restores this data backup into the specified slave replica.





This procedure can be initiated by specifying the database unit (PLDB or a DSG) and the target node for the restoration.

The command generates backup files, copies the files to the target node, and removes the backup files at the end of the restore procedure.

For further information, refer to *CUDB Backup and Restore Procedures*, Reference [11] and *CUDB Node Commands and Parameters*, Reference [9].

### 2.3.7.9 Automated Procedure for Complete CUDB System Backup and Restore

The `cudbSystemDataBackupAndRestore` command automates the backup and restore procedure for a complete CUDB system. In this backup type all information stored in CUDB is affected. A CUDB system data backup is composed of single backups of each individual database unit inside CUDB.

The following have to be considered when performing a system data backup and restore:

- For complete data backup, see Section 2.3.7.2 on page 14.
- If consistency between PLDB and DSG parts of the system data backup is preferred to continuous provisioning, provisioning operations must be stopped during the CUDB system data backup process.
- System data restore can only be executed on a system that is exactly the same as the system from which the backup was taken.
- Simplified system data restore works for a fully operating system only.

For further information, refer to *CUDB Backup and Restore Procedures*, Reference [11] and *CUDB Node Commands and Parameters*, Reference [9].

### 2.3.8 Detecting Inconsistencies between Replicas

Due to the nature of asynchronous replication, if there is a replication lag between the master DS unit and the slave DS unit of a DSG, the data content of the master and the slave is different at any given time. This “temporary” inconsistency is normal.

However, under normal operational conditions the replication lag is small and causes only minimal data content difference. Based on this assumption, CUDB offers a solution that performs lightweight replica consistency checking based on the difference of tables size in master and slave replicas.

The feature checks the PLDB and all DSG slave replicas at configurable intervals, and it can also be executed manually using the `cudbCheckConsistency` command. For more information on this command, see *CUDB Node Commands and Parameters*, Reference [9].



Inconsistencies can also be introduced by other factors, such as replication channel problems caused by network disturbances, or by human errors (for example inappropriate backup handling or maintenance operation). To handle such inconsistencies, CUDB offers a solution called **Consistency Check**, which performs a consistency check that can detect inconsistency between master and slave replicas by performing a scan of the object classes and attributes in LDAP entries. For more information about Consistency Check, see *CUDB Consistency Check*, Reference [12].

The `cudbConsistencyMgr` command can order consistency tasks to check that a master and a slave replica of the same DSG or PLDB are consistent. It is also possible to list each pending and running consistency check task within all sites of the CUDB system. For more information on this command, refer to *CUDB Node Commands and Parameters*, Reference [9].

#### 2.3.8.1 Consistency Check Output Files

The Consistency Check tasks produce several log files, which contain information about the task execution and the found inconsistencies. For more information about these logs, refer to *CUDB Consistency Check*, Reference [12].

### 2.3.9 Automatic Handling of Network Isolation

This section contains information about the Automatic Handling of Network Isolation function in CUDB.

The Automatic Handling of Network Isolation function maximizes service availability for network isolation scenarios, therefore removing the potential impact of partial loss of service in an isolated site or set of sites during a network incident. The function ensures that in case of an even symmetrical split situation, the two halves of the system always provide service: this is achieved by making the CUDB system working in a "multi-master" mode during network incidents. In this mode, both halves execute changes, but they cannot replicate their local changes to the other half, therefore temporarily behaving as two diverging databases. When the network is recovered, Automatic Handling of Network Isolation ensures that all the modifications executed during the "multi-master" mode are automatically identified, merged together into the master (that is, "repaired") and kept.

During the execution of Automatic Handling of Network Isolation, the Selective Replica Check and then the Data Repair processes are run to achieve the final result.

#### 2.3.9.1 Selective Replica Check

The Selective Replica Check process is started on a database cluster, which was formerly a master, but is now a slave, and is unable to sync with the current master. It tries identifying the changes on the database cluster that may



have not been replicated to the current master (a side effect of asynchronous replication) by reading the operational logs of that database cluster in the payload blades or VMs.

### 2.3.9.2 Data Repair

The Data Repair process is started as part of Automatic Handling of Network Isolation if and when Selective Replica Check has been completed.

Data Repair takes the Selective Replica Check output as its input, containing information about the LDAP entries. For each LDAP entry, the timestamp of the last update on the former master is recorded, together with the type of operation and the latest contents of the entry on the former master.

Data Repair compares the LDAP entries in the output of Selective Replica Check with data in the current master replica. Then, based on this information, on an entry-by-entry basis, Data Repair decides whether to keep the data in the current master replica as it is or to update/delete the LDAP entry on the current master replica, based on the information from the former master replica. Each entry in the input is logged to the Data Repair output logs, either to the `repaired` entries log or the `unrepaired` entries log.

The corresponding alarms are raised to inform the operator about repaired and/or unrepaired entries.

### 2.3.9.3 Logs

For the format and interpretation of the output logs of the Automatic Handling of Network Isolation processes, refer to *CUDB Automatic Handling of Network Isolation Output Description*, Reference [13].

### 2.3.10 Self-Ordered Backup and Restore

The Self-Ordered Backup and Restore process is started on a slave database cluster if it is unable to sync with the current master. The process will try to recover the original geographical redundancy configuration, 1+1 double or 1+1+1 triple geographical redundancy, by creating an individual unit data backup on the current master PLDB or DSG replica and restore that unit data backup on its local slave replica without any intervention.

## 2.4 Dependencies and Interactions

This section provides information on the various dependencies and interactions.



### 2.4.1 Consistency Check Interactions

The Consistency Check task does not start, and the ongoing ones stop, if any of the following conditions are met:

- Either of the compared replicas:
  - becomes inaccessible.
  - becomes masterless.
  - becomes degraded.
  - is put into maintenance mode.
  - is stopped.
- The replication lag exceeds the specified limit.
- Mastership change occurs.
- Unit data restore is performed on the checked slave.
- Replication is down.
- Information on replication lag is unavailable.

A new task can be started for other non-degraded DSs if an ongoing task was interrupted due to degradation. A new task does not start on the degraded DS.

It is possible to stop an ongoing replica consistency check task manually during the execution, or to remove it from the Pending Task List.

The root cause of the stopped or failed replica consistency check tasks is logged locally in the given site, which can be checked with the `cudbConsistencyMgr -t` command.

During a consistency check task, it is not allowed to perform any LDAP schema change. It is not allowed to run consistency check task during an upgrade.

The consistency check process tries to minimize the impact on the traffic handling capacity of the CUDB system. It may cause some bandwidth usage increase.

### 2.4.2 Backup Interactions

In general, system-level and unit-level backup operations do not necessarily interact with other CUDB system functions. However, when a PLDB or DS replica is backed up, negative performance impact is expected due to the constant information flow between the memory and the local disk storage system at every blade or VM involved in the backup. If the backed up replica is a master replica, the performance impact also affects the number of traffic



operations in each unit that the CUDB system can handle at a time. At the same time, the backup of the DSG master replica only impacts the traffic operations that are directed to the set of subscribers allocated to the DSG that the master replica belongs to.

It is strongly recommended to observe the following rules for backup executions:

- Avoid executing system data backups right after a unit data restore of a DSG slave replica. Since the DSG slave replica is possibly still synchronizing with its master, a system data backup performed right after a unit data restore can result in inconsistencies if that replica is chosen as the source of the individual DSG data backup.
- Avoid executing unit data backups during major provisioning operations, as, due to provisioning, the backup would soon become obsolete.
- Try to schedule periodic system data backups outside any high update traffic time.
- Never execute system data backups while high update traffic is served by the system.

The execution of a backup does not affect regular traffic operations and traffic operations do not affect the backup. Provisioning might have minor impacts on system data backups. For more information, refer to *CUDB Backup and Restore Procedures*, Reference [11].

### 2.4.3 Restore Interactions

During the execution of a PLDB or DSG unit data restore, the PLDB and DSG replicas affected by the restoration are unavailable. The PLDB and DSG replica downtime ends when the restore process is successfully finished. The downtime of a PLDB replica results in making its whole node becoming unavailable. Refer to *CUDB High Availability*, Reference [4] for more information.

During the execution of a DSG group data restore, the affected DSG is unavailable as all replicas are down at the same time, and does not become available until the individual unit data restore operation successfully finishes in at least one of the replicas, and a new master replica for that DSG is selected. After a group data restore in a DSG, a reconciliation process takes place in that DSG. See Section 2.3.6 on page 9 for reconciliation details.

During a PLDB group data restore, the PLDB is unavailable as all replicas are down at the same time. The PLDB does not become available until the unit data restore operation successfully finishes in at least one of the replicas, and a new master replica for the PLDB is selected. As an unavailable PLDB replica takes down the whole node where it is hosted, a PLDB group data restore takes down the whole CUDB system, therefore no service is provided. CUDB nodes come back online as restore operations for the PLDB finish. Master replicas for DSGs are chosen as CUDB nodes come back online and DSG replicas become available. After a PLDB group data restore, a reconciliation



process is started for all DSGs in the system. See Section 2.3.6 on page 9 for reconciliation details.

During the execution of a system data restore, the entire CUDB system is unavailable, therefore no service is provided at all. Once all the individual unit data restore processes finish, new master selections take place, and the system gradually starts providing service again. See *CUDB High Availability*, Reference [4] for more information.

## 2.4.4 Provisioning Assurance Interactions

If the Provisioning Assurance feature is configured in the CUDB system, and a Provisioning Assurance replay is taking place, the execution of Selective Replica Check, Data Repair and reconciliation tasks will be delayed. These tasks will start when the Provisioning Gateway finishes the reprovisioning operations. For configuring the Provisioning Assurance feature, refer to *CUDB High Availability*, Reference [4].

**Note:** A mastership change in DSG  $x$  will interrupt an ongoing reconciliation task on DSG  $x$ , irrespective of whether Provisioning Assurance is enabled or not.

## 2.4.5 Automatic Handling of Network Isolation Interactions

If the Selective Replica Check and Data Repair processes are enabled, the interactions in the following subsections apply. Besides, reconciliation will not be automatically triggered after group data restore for a DSG or PLDB, and after system data restore.

### 2.4.5.1 Selective Replica Check Interactions

Selective Replica Check will not start if the Provisioning Assurance task is running, and it will only start on the database cluster if it is in ready mode.

Selective Replica Check can run even if the local PLDB replica is down or the node does not have a local PLDB. If there are any other issues that would prevent LDAP queries on the database cluster where Selective Replica Check is started, the Selective Replica Check execution will fail. Do not launch the Backup and Restore activity manually on the database cluster where Selective Replica Check runs, only after the Selective Replica Check is finished and the results are properly logged.

**Note:** If Backup and Restore is started in any way, it will destroy the existing operational logs which are required by Selective Replica Check.

Execution of an ongoing Selective Replica Check process is gracefully stopped if it exceeds a 24 hour time limit.



### 2.4.5.2 Data Repair Interactions

The following considerations apply to Data Repair interactions:

- Data Repair performs write operations through LDAP. Therefore, it interacts with other write operations just like traffic or provisioning. If reallocation is performed for an LDAP entry that is being repaired, the repair might fail and the entry would be recorded in the `unrepaired` log.
- Data Repair performs LDAP operations on the current PLDB or DSG master. If the PLDB or DSG mastership is changed during the repair process, Data Repair will continue accessing the newly chosen master replica.
- Consecutive mastership changes in the same data partition will result into Selective Replica Check and Data Repair being triggered at both former master replicas. Execution of these two processes is independent and data repair operations might interfere with each other.
- If Data Repair is being executed on PLDB and DSGs at the same time, repair-delete operations in PLDB might fail if the matching repair-delete operations in a DSG have not been executed yet.
- Execution of an ongoing Data Repair process is gracefully stopped if it exceeds a 24 hour time limit decreased by the execution time of the preceding Selective Replica Check.

### 2.4.6 Self-Ordered Backup and Restore Interactions

In case the Automatic Handling of Network Isolation function is enabled, the Self-Ordered Backup and Restore process is started if and when a Selective Replica Check subtask has been completed, irrespective of its outcome. The Data Repair and Self-Ordered Backup and Restore tasks run parallel without conflict.

The Self-Ordered Backup and Restore process creates an individual unit data backup on the current master PLDB or DSG replica, and restores it on its local slave replica. For more information about backup and restore interactions, refer to Section 2.4.2 on page 24 and Section 2.4.3 on page 25.

Manual Backup and Restore activity will fail on a database cluster if Self-Ordered Backup and Restore is already running.







## 3 Operation and Maintenance

This section provides information related to the operation and maintenance of the data storage handling features of CUDB.

### 3.1 Configuration

This section provides information on the configuration of the CUDB data storage and restore features.

#### 3.1.1 Data Store Configuration

Data store management is configured through model-driven actions over the CUDB configuration model. In this model, each CUDB node reflects both of its own detailed configuration, along with an overview of the rest of the CUDB nodes in the system through the `CudbLocalNode` and `CudbRemoteNode` Management Object Classes, respectively. This model structure is the starting point of the data store configuration in CUDB. Refer to *CUDB Node Configuration Data Model Description*, Reference [6] for more details on the configuration model.

#### 3.1.2 Data Storage Parameters

As part of the CUDB internal configuration model, the following Management Object Classes are used to manage the DSs:

- System-level object classes: `CudbDsGroup` and `CudbPlGroup`.
- Object classes for managing local nodes: `CudbLocalPl` and `CudbLocalDs`.
- Object classes for managing remote nodes: `CudbRemotePl` and `CudbRemoteDs`.

Refer to *CUDB Node Configuration Data Model Description* for more details about the CUDB internal configuration model.

#### 3.1.3 System Restore Configuration

Following a successful system data restore, the manual restore of the stored SQL procedures is required. This process requires a proper username and password to access the database clusters needed for the successful SQL procedure restore. Refer to *CUDB Backup and Restore Procedures*, Reference [6] for more information on recovering stored SQL procedures, and *CUDB*



*Security and Privacy Management*, Reference [24] for more information on usernames and passwords.

### 3.1.4 Miscellaneous Tasks

This section describes miscellaneous tasks related to the operation and maintenance of the data storage handling features.

#### 3.1.4.1 Data Storage Tasks

The miscellaneous data storage tasks associated with data storage handling include the following:

- **Defining a new site**

New sites can be defined as part of a system enlargement with the aim to accommodate extra nodes in the system. The new site information must be configured in the system-wide information present in all nodes. This procedure can be performed by Ericsson personnel only. Contact the next level of maintenance support, in case the system needs to be expanded.

- **Introducing a node in the CUDB system**

A CUDB node can be introduced in a CUDB system to provide extra capacity, or as a result of a previous isolation for maintenance reasons. This procedure can be performed by Ericsson personnel only. Contact the next level of maintenance support, in case the system needs to be expanded.

- **Creating a DSG**

A management procedure is available in CUDB for defining a new DSG in an existing CUDB system, in case the storage needs of the system require to do so. The new DSG group must be defined in each node of the system which is targeted to be part of the DSG.

Refer to *CUDB System Administrator Guide*, Reference [10] for more details about management steps to consider.

**Note:** The DSG creation is the first step prior to the actual activation of the DSG. The activation leaves the system in a state in which the new DSG is ready for data provisioning.

- **Activating a DSG**

In order to activate the DSG for the whole CUDB system, its data partitions (that is, DS Units) must be activated in each node sharing the DSG. The DSG activation can happen either if the specific DSG was newly created, or if it was previously deactivated.

The order of DS Unit activation in this case is as follows: first, the DS Unit planned to be the master of the DSG must be activated in the



corresponding node. Then, the slave DS units (one or two, depending on the data redundancy scheme) must be activated in the rest of the nodes which form the system.

For details of the management procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Adding a DS Unit in the CUDB node**

In case a new DSG is created in the CUDB system, the corresponding DS Units must also be created in each node assigned to the new DSG.

As an obvious prerequisite, the infrastructure hosting the DS Units in the node must be installed. This means two blades or VMs for each DS Unit to maintain redundancy.

Once the blades or VMs are installed, the CUDB software (including the LDE and SAF) must be loaded and configured, after which the activation of the local and remote DS Units can proceed (see the previous task about DS Unit activation).

For details of the management procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Activating a DS Unit**

Once the DSGs are defined in the CUDB system, the DS Units being part of the DSGs must be activated in the CUDB configuration model, so they can become operational both in the local nodes and the remote nodes of the CUDB system.

For details of the management procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Deactivating a DS Unit**

At the cost of master eligibility inside the DSG it belongs to, a DS Unit can be deactivated. Deactivation is carried out through the related command of the CUDB configuration model. For details of the procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Activating a PLDB Unit**

The PLDB must be activated in the CUDB configuration model to become operational. In each CUDB node, such activation consists of enabling both the local and remote representation (from remote nodes in the system) in the configuration model.

For details of the procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Deactivating a PLDB Unit**



At the cost of master eligibility inside the PL, a PLDB instance in a CUDB node can be deactivated. Deactivation is carried out through the related command of the CUDB configuration model. For details of the procedure to follow, refer to *CUDB System Administrator Guide*, Reference [10].

- **Replacing a blade or recovering a VM**

Blades or VMs that comprise the storage component nodes can be replaced or recovered, respectively, without CUDB service interruption. To guarantee successful replacement or recovery, a specific procedure must be followed. For further information on these procedures, refer to *Server Platform, Blade Replacement*, Reference [25] for CUDB systems deployed on native BSP 8100, or to *Virtualized CUDB Virtual Machine Recovery*, Reference [26] for CUDB systems deployed on a cloud infrastructure.

- **Setting a PLDB cluster in maintenance mode**

The PLDB can be set to maintenance mode, which effectively takes the CUDB node out of service. Maintenance mode is initiated with the `cudbManageStore` command. Refer to *CUDB Node Commands and Parameters*, Reference [9] for more information on the procedure.

- **Setting a DS Unit in maintenance mode**

A DS Unit can be set to maintenance mode. In case the DS Unit in question acts as the master partition in its DSG, a mastership change results from this operation. Maintenance mode is initiated with the `cudbManageStore` command. Refer to *CUDB Node Commands and Parameters*, Reference [9] for more information on the procedure.

### 3.1.4.2 Backup and Restore Tasks

The tasks related to the backup and restore procedures are as follows:

- **Bringing a slave replica up-to-date with its master replica**

As explained in *CUDB High Availability*, Reference [4], slave replicas are kept synchronized with their masters by means of an asynchronous replication protocol. Under certain conditions (like a mastership change) the slave replica can lag behind its master up to a point where automatic synchronization is not feasible anymore. If this situation arises, a backup must be created from the master replica, and it must be used to perform a unit restore in the slave replicas that are lagging behind.

This task is based on unit backups and unit restores. For this purpose, the `cudbUnitDataBackupAndRestore` command can be used. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information on the procedure.

- **Recovering a failed replica**



Certain infrastructure failures can render a replica unable to recover by itself once the failure is fixed. This situation arises when the whole storage system of the blades or VMs that run the replica fails simultaneously. In this case, all persistent data stored by the replica is lost, and the replica has to be recovered from a unit backup of the current master replica.

This task is based on unit backups and unit restores. For this purpose, the `cudbUnitDataBackupAndRestore` command can be used. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information on the procedure.

- **Recovering a failed DSG**

When multiple infrastructure components fail simultaneously, the entire DSG can fail, resulting in losing all the data it contains. When this happens, all data stored by the DSG becomes unavailable (refer to *CUDB High Availability*, Reference [4] for more information). In order to restore access to those data, it is necessary to make the DSG operational again by using the most recent backup available, be it either a unit backup, or a full system backup.

**Note:** In case the complete DSG fails, all data added to that DSG since the last backup will be lost, and it will be impossible to recover them. However, failure of a complete DSG is an extremely unlikely event, and is never supposed to happen.

Recovering a failed DSG involves performing a group restore. Refer to *CUDB Backup and Restore Procedures*, Reference [11] for more information on the procedure. Inconsistencies detected after the DSG recovery are solved automatically by the Reconciliation feature. See Section 2.3.6 on page 9 for more information.

- **Backing up a unit**

For a detailed step-by-step information on how to backup a unit (either a PLDB or DSG database replica), refer to *CUDB Backup and Restore Procedures*, Reference [11].

- **Backing up the CUDB system**

For a detailed step-by-step information on how to backup a complete CUDB system, refer to *CUDB Backup and Restore Procedures*, Reference [11].

- **Scheduling a periodic system backup**

Periodic system backups must be scheduled on every CUDB node according to the steps detailed in *CUDB Backup and Restore Procedures*, Reference [11].

- **Restoring a unit**

For a detailed step-by-step information on how to restore a unit, refer to *CUDB Backup and Restore Procedures*, Reference [11].



- **Restoring a PLDB**

Restoration of a PLDB is only possible by restoring the entire CUDB system, refer to *CUDB Backup and Restore Procedures*, Reference [11].

- **Restoring a group**

For a detailed step-by-step information on how to restore a whole group, refer to *CUDB Backup and Restore Procedures*, Reference [11].

- **Restoring the CUDB system**

For a detailed step-by-step information on how to restore the entire CUDB system, refer to *CUDB Backup and Restore Procedures*, Reference [11].

### 3.1.5 Consistency Check Configuration

Consistency Check tasks have configurable parameters which can be specified with the `cudbConsistencyMgr` command. For more information about `cudbConsistencyMgr` command, refer to *CUDB Node Commands and Parameters*, Reference [9].

## 3.2 Fault Management

This section provides detailed information about the fault management of the CUDB data handling features.

### 3.2.1 Data Storage Management Alarms

The CUDB system can raise the following alarms during data storage management:

- `DS cluster down`

For more information, refer to *Storage Engine, DS Cluster Down*, Reference [27].

- `PLDB cluster down`

For more information, refer to *Storage Engine, PLDB Cluster Down*, Reference [28].

- `DS cluster in maintenance mode`

For more information, refer to *Storage Engine, DS Cluster in Maintenance Mode*, Reference [29].

- `PLDB cluster in maintenance mode`



For more information, refer to *Storage Engine, PLDB Cluster in Maintenance Mode*, Reference [30].

- Memory usage too high in DS, Warning Threshold Reached

For more information, refer to *Storage Engine, Memory Usage Too High In DS, Warning Threshold Reached*, Reference [31].

- Memory usage too high in DS, Full Threshold Reached

For more information, refer to *Storage Engine, Memory Usage Too High In DS, Full Threshold Reached*, Reference [32].

- Out of memory in DS

For more information, refer to *Storage Engine, Out Of Memory In DS*, Reference [33].

- Memory usage too high in PLDB, Warning

For more information, refer to *Storage Engine, Memory Usage Too High In PLDB, Warning*, Reference [34].

- Memory usage too high in PLDB, Major

For more information, refer to *Storage Engine, Memory Usage Too High In PLDB, Major*, Reference [35].

- Out of memory in PLDB

For more information, refer to *Storage Engine, Out Of Memory In PLDB*, Reference [36].

- High Load in DS

For more information, refer to *Storage Engine, High Load In DS*, Reference [37].

- High Load in PLDB

For more information, refer to *Storage Engine, High Load In PLDB*, Reference [38].

### 3.2.2 Backup Alarms

The CUDB system can raise the following alarms when performing a backup operation:

- Storage Engine, Backup Fault In PLDB

For more information, refer to *Storage Engine, Backup Fault In PLDB*, Reference [39].



- Storage Engine, Backup Fault in DS

For more information, refer to *Storage Engine, Backup Fault In DS*, Reference [40].

- Storage Engine, Backup Notification Failure to Provisioning Gateway

For more information, refer to *Storage Engine, Backup Notification Failure to Provisioning Gateway*, Reference [41].

### 3.2.3 Restore Alarms

The CUDB system can raise the following alarms when performing a restore operation:

- Storage Engine, Restore Fault in PLDB

For more information, refer to *Storage Engine, Restore Fault in PLDB*, Reference [42].

- Storage Engine, Restore Fault in DS

For more information, refer to *Storage Engine, Restore Fault in DS*, Reference [43].

### 3.2.4 Reconciliation Alarms

The CUDB system can raise the following alarms when performing a restore operation:

- Storage Engine, Temporary Data Inconsistency

For more information, refer to *Storage Engine, Temporary Data Inconsistency*, Reference [44].

- Storage Engine, Deleted Data Due to Reconciliation

For more information, refer to *Storage Engine, Deleted Data Due to Reconciliation*, Reference [45].

### 3.2.5 Replica Inconsistency Alarms

The CUDB system can raise the following replica inconsistency related alarms:

- Storage Engine (PLDB), Potential data inconsistency between replicas found

For more information, refer to *Storage Engine, Potential Data Inconsistency between Replicas Found in PLDB*, Reference [46].





- Storage Engine (DS-group #dsg), Potential data inconsistency between replicas found

For more information, refer to *Storage Engine, Potential Data Inconsistency between Replicas Found in DS*, Reference [47].

- Storage Engine, Data Inconsistency Between Replicas Found in DS, Minor

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Found in DS, Minor*, Reference [48].

- Storage Engine, Data Inconsistency Between Replicas Found in DS, Major

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Found in DS, Major*, Reference [49].

- Storage Engine, Data Inconsistency Between Replicas Found in PLDB, Minor

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Found in PLDB, Minor*, Reference [50].

- Storage Engine, Data Inconsistency Between Replicas Found in PLDB, Major

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Found in PLDB, Major*, Reference [51].

- Storage Engine, Data Inconsistency Between Replicas Repaired, PLDB

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Repaired, PLDB*, Reference [16].

- Storage Engine, Data Inconsistency Between Replicas Repaired, DS

For more information, refer to *Storage Engine, Data Inconsistency between Replicas Repaired, DS*, Reference [17].

- Storage Engine, Unrepaired Data Inconsistency Between Replicas, PLDB

For more information, refer to *Storage Engine, Unrepaired Data Inconsistency between Replicas, PLDB*, Reference [18].

- Storage Engine, Unrepaired Data Inconsistency Between Replicas, DS

For more information, refer to *Storage Engine, Unrepaired Data Inconsistency between Replicas, DS*, Reference [19].



- Storage Engine, Execution of Selective Replica Check Failed, PLDB, Major

For more information, refer to *Storage Engine, Execution of Selective Replica Check Failed, PLDB, Major*, Reference [14].

- Storage Engine, Execution of Selective Replica Check Failed, DS, Major

For more information, refer to *Storage Engine, Execution of Selective Replica Check Failed, DS, Major*, Reference [15].

### 3.2.6 Replication Alarms

The CUDB system can raise the following alarms related to replication channel state:

- Storage Engine, Replication Stopped Working in PLDB

For more information, refer to *Storage Engine, Replication Stopped Working in PLDB*, Reference [52].

- Storage Engine, Replication Stopped Working in DS

For more information, refer to *Storage Engine, Replication Stopped Working in DS*, Reference [53].

- Storage Engine, Replication Channels Down in DS

For more information, refer to *Storage Engine, Replication Channels Down in DS*, Reference [54].

- Storage Engine, Replication Channels Down in PLDB

For more information, refer to *Storage Engine, Replication Channels Down in PLDB*, Reference [55].

- Storage Engine, Replication Delay Too High In DS

For more information, refer to *Storage Engine, Replication Delay Too High In DS*, Reference [56].

- Storage Engine, Replication Delay Too High In PLDB

For more information, refer to *Storage Engine, Replication Delay Too High In PLDB*, Reference [57].

- Storage Engine, Unable to Synchronize Cluster in DS, Major

For more information, refer to *Storage Engine, Unable to Synchronize Cluster in DS, Major*, Reference [22].



- Storage Engine, Unable to Synchronize Cluster in PLDB, Major

For more information, refer to *Storage Engine, Unable to Synchronize Cluster in PLDB, Major*, Reference [23].

- Storage Engine, Unable to Synchronize Cluster in DS, Warning

For more information, refer to *Storage Engine, Unable to Synchronize Cluster in DS, Warning*, Reference [20].

- Storage Engine, Unable to Synchronize Cluster in PLDB, Warning

For more information, refer to *Storage Engine, Unable to Synchronize Cluster in PLDB, Warning*, Reference [21].

- Storage Engine, Automatic Handling of Network Isolation not Completed for DS

For more information, refer to *Storage Engine, Automatic Handling of Network Isolation not Completed for DS*, Reference [58].

- Storage Engine, Automatic Handling of Network Isolation not Completed for PLDB

For more information, refer to *Storage Engine, Automatic Handling of Network Isolation not Completed for PLDB*, Reference [59].

### 3.2.7 Troubleshooting

Refer to *CUDB Troubleshooting Guide*, Reference [60] for more information on fault management.

## 3.3 Performance Management

This section provides detailed information on the performance management services related to the data storage handling features.

### Data Storage Management Counters

The following cumulative counters are associated to data storage management:

- PLDB cluster counters group: This set of counters provides information about the local PLDB replica.
- DS cluster counters group: These sets of counters provide information about each of the local DS units.



For more information, refer to *CUDB Counters List*, Reference [61].

## 3.4 Security

This section contains information on the security measures associated with the data storage handling features of CUDB.

### Automatic Handling of Network Isolation

Root permission or `cudbadmin` group membership is needed to access the Automatic Handling of Network Isolation output logs.

### Backup and Restore Security Management

The following security-related information applies to the backup and restore procedures:

<b>Access Control</b>	Only <code>root</code> or users in <code>cudbadmin</code> group can access the backup files.
<b>Protocols</b>	Backup and restore procedures are executed through Secure Shell (SSH) sessions. The outcome of these operations is also checked through this protocol.
<b>Roles</b>	Backup and restore procedures in CUDB can only be executed by users with system administration privileges. The system administration privileges are also required to read and write to the persistent disk storage system of the blades or VMs, as well as to the NFS storage of the CUDB nodes.
<b>User authentication</b>	The user setting up an SSH session to the CUDB system is authenticated by means of a username and a password.
<b>Media</b>	Files generated during unit backups and system backups are neither ciphered, nor encrypted. When safe-keeping those files to media external to the CUDB system, care must be taken to prevent unauthorized parties accessing the files.

### Consistency Check

Root permission is required to run consistency check. Users belonging to the `cudbadmin` group can run it through `sudo` (without password).



## Export Security Management

**Access Control** Only `root` or users in `cudbadmin` group can access the export files.

## 3.5 Logging

This section provides information on the logging services related to the data storage handling features.

### 3.5.1 Backup and Restore Log Management

In addition to the related specific logs and alarms, the execution of backup and restore commands also generate console output containing useful information for the further management of backup files. This section only describes the most relevant output in more detail: the system data backup output.

#### System Data Backup Standard Output

When executing a system backup, the following information is produced through the standard output channel of the SSH session, over which the backup order was initiated:

- The version of the backup SW.
- The time and date when the backup was completed, as recorded in the filenames of the TAR files containing backup data.
- The units where the individual unit backups were executed successfully.
- The units where the individual unit backups failed (in case of failures).
- The CUDB nodes where the above-mentioned units belong.

Example 1 shows how the standard output of a system data backup looks like on the shell where the backup command was initiated. The CUDB system for the example below consists of three nodes, where the PLDB master replica resides in Node 1 and runs on four blades or VMs, while the DSG replicas selected for backup reside in Nodes 1-3 respectively, and each of them run on two blades or VMs.



```
cudb_backup ver(1.2.18)
BEGIN MANAGEMENT FOR LAST BACKUP
Backup 2010-02-24_11-42 finished successfully in :

PLDB in CUDB node 1
NDB node PL_2_3
NDB node PL_2_4
NDB node PL_2_5
NDB node PL_2_6
DSG#1 in CUDB node 1
NDB node PL_2_7
NDB node PL_2_8
DSG#2 in CUDB node 2
NDB node PL_2_9
NDB node PL_2_10
DSG#3 in CUDB node 3
NDB node PL_2_11
NDB node PL_2_12
```

*Example 1 Example Output of System Data Backup*

**Note:** The PL\_x\_y resources in the output above are the names of the payload blades or VMs running the units.

Based on the output above, the NFS media of Node 1 must contain six files (one on each blade or VM for the PLDB master replica, and one on each blade or VM for the DSG1 replica residing in Node 1). An example of the file names can be found below.

**Note:** PLDB backup file names carry number 0 as the DSG ID value.

**PLDB backup files:**

```
BACKUP-2010-02-24_11-42-1.0.3.tar
BACKUP-2010-02-24_11-42-1.0.4.tar
BACKUP-2010-02-24_11-42-1.0.5.tar
BACKUP-2010-02-24_11-42-1.0.6.tar
```

**DSG1 replica backup files:**

```
BACKUP-2010-02-24_11-42-2.1.3.tar
BACKUP-2010-02-24_11-42-2.1.4.tar
```

At the same time, the NFS media of Node 2 contains two files (one on each blade or VM for the DSG2 replica residing in Node 2). An example of the file names can be found below.

**DSG2 replica backup files:**

```
BACKUP-2010-02-24_11-42-3.2.3.tar
BACKUP-2010-02-24_11-42-3.2.4.tar
```

Finally, the NFS media of Node 3 contains two files as well (one on each blade or VM for the DSG3 replica residing in Node 3). An example of the file names can be found below.

**DSG3 replica backup files:**`BACKUP-2010-02-24_11-42-1.3.3.tar``BACKUP-2010-02-24_11-42-1.3.4.tar`**3.5.2 Reconciliation Log Management**

The reconciliation procedure logs the following events:

- The DS reconciliation process initiation. The event is indicated with the following message:

```
(info) - Reconciliation Process for <dsgId> Started.
```

- The DS reconciliation process termination. The event is indicated with the following message:

```
(info) - Reconciliation Process for <dsgId> Finished.
```

- The removal of each inconsistency entry found in the DS reconciliation process. The event is indicated with the following message:

```
(warning) - Deleted Entry due to Reconciliation  
Process: [ <mscId | assocId> ].
```

For further information about the logs provided by this procedure, refer to *CUDB Node Logging Events*, Reference [62].







## Glossary

For the terms, definitions, acronyms and abbreviations used in this document, refer to *CUDB Glossary of Terms and Acronyms*, Reference [63].





## Reference List

### CUDB Documents

- [1] *CUDB LDAP Interwork Description*
- [2] *CUDB Technical Product Description*
- [3] *CUDB Data Distribution*
- [4] *CUDB High Availability*
- [5] *CUDB Binary Large Object Attributes Management*
- [6] *CUDB Node Configuration Data Model Description*
- [7] *CUDB Subscription Reallocation*
- [8] *CUDB Multiple Geographical Areas*
- [9] *CUDB Node Commands and Parameters*
- [10] *CUDB System Administrator Guide*
- [11] *CUDB Backup and Restore Procedures*
- [12] *CUDB Consistency Check*
- [13] *CUDB Automatic Handling of Network Isolation Output Description*
- [14] *Storage Engine, Execution of Selective Replica Check Failed, PLDB, Major*
- [15] *Storage Engine, Execution of Selective Replica Check Failed, DS, Major*
- [16] *Storage Engine, Data Inconsistency between Replicas Repaired, PLDB*
- [17] *Storage Engine, Data Inconsistency between Replicas Repaired, DS*
- [18] *Storage Engine, Unrepaired Data Inconsistency between Replicas, PLDB*
- [19] *Storage Engine, Unrepaired Data Inconsistency between Replicas, DS*
- [20] *Storage Engine, Unable to Synchronize Cluster in DS, Warning*
- [21] *Storage Engine, Unable to Synchronize Cluster in PLDB, Warning*
- [22] *Storage Engine, Unable to Synchronize Cluster in DS, Major*
- [23] *Storage Engine, Unable to Synchronize Cluster in PLDB, Major*



- [24] *CUDB Security and Privacy Management*
- [25] *Server Platform, Blade Replacement*
- [26] *Virtualized CUDB Virtual Machine Recovery*
- [27] *Storage Engine, DS Cluster Down*
- [28] *Storage Engine, PLDB Cluster Down*
- [29] *Storage Engine, DS Cluster in Maintenance Mode*
- [30] *Storage Engine, PLDB Cluster in Maintenance Mode*
- [31] *Storage Engine, Memory Usage Too High In DS, Warning Threshold Reached*
- [32] *Storage Engine, Memory Usage Too High In DS, Full Threshold Reached*
- [33] *Storage Engine, Out Of Memory In DS*
- [34] *Storage Engine, Memory Usage Too High In PLDB, Warning*
- [35] *Storage Engine, Memory Usage Too High In PLDB, Major*
- [36] *Storage Engine, Out Of Memory In PLDB*
- [37] *Storage Engine, High Load In DS*
- [38] *Storage Engine, High Load In PLDB*
- [39] *Storage Engine, Backup Fault In PLDB*
- [40] *Storage Engine, Backup Fault In DS*
- [41] *Storage Engine, Backup Notification Failure to Provisioning Gateway*
- [42] *Storage Engine, Restore Fault in PLDB*
- [43] *Storage Engine, Restore Fault in DS*
- [44] *Storage Engine, Temporary Data Inconsistency*
- [45] *Storage Engine, Deleted Data Due to Reconciliation*
- [46] *Storage Engine, Potential Data Inconsistency between Replicas Found in PLDB*
- [47] *Storage Engine, Potential Data Inconsistency between Replicas Found in DS*
- [48] *Storage Engine, Data Inconsistency between Replicas Found in DS, Minor*



- [49] *Storage Engine, Data Inconsistency between Replicas Found in DS, Major*
- [50] *Storage Engine, Data Inconsistency between Replicas Found in PLDB, Minor*
- [51] *Storage Engine, Data Inconsistency between Replicas Found in PLDB, Major*
- [52] *Storage Engine, Replication Stopped Working in PLDB*
- [53] *Storage Engine, Replication Stopped Working in DS*
- [54] *Storage Engine, Replication Channels Down in DS*
- [55] *Storage Engine, Replication Channels Down in PLDB*
- [56] *Storage Engine, Replication Delay Too High In DS*
- [57] *Storage Engine, Replication Delay Too High In PLDB*
- [58] *Storage Engine, Automatic Handling of Network Isolation not Completed for DS*
- [59] *Storage Engine, Automatic Handling of Network Isolation not Completed for PLDB*
- [60] *CUDB Troubleshooting Guide*
- [61] *CUDB Counters List*
- [62] *CUDB Node Logging Events*
- [63] *CUDB Glossary of Terms and Acronyms*