

CUDB Consistency Check

User Guide

Copyright

© Ericsson AB 2016-2018. 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

1	Introduction	1
1.1	Document Purpose and Scope	1
1.2	Target Group	1
1.3	Prerequisites	1
1.4	Revision Information	1
1.5	Typographic Conventions	2
2	Overview	3
2.1	Execution Conditions	4
3	Consistency Check	6
3.1	Ordering the Consistency Check	6
3.2	Listing Tasks	7
3.3	Reviewing the Task History	8
3.4	Removing Tasks	9
3.5	Consistency Check Output	10
3.6	Alarms	18
3.7	Use Recommendations	19
	Glossary	21
	Reference List	22





1 Introduction

This document describes how the Ericsson Centralized User Database (CUDB) Consistency Check function is used to find the differences between a slave replica and the corresponding master replica of a Data Store Unit Group (DSG) or the Processing Layer Database (PLDB).

1.1 Document Purpose and Scope

This document provides a guide to use CUDB Consistency Check.

1.2 Target Group

Users of this document must be CUDB system administrators with knowledge of the CUDB system and the Lightweight Directory Access Protocol (LDAP).

1.3 Prerequisites

Before performing any of the operations described in this document, make sure that the conditions for running the tool are met. See [Execution Conditions](#) on page 4 for more information.

1.4 Revision Information

Rev. A

This document is based on 17/1553-CSH 109 067/9 with the following changes:

- Terminology updates throughout the document because of virtualized deployment support.
- Updated the default value of the maxReplicaLag parameter throughout the document.
- [Execution Conditions](#) on page 4: Updated the administrative command.

Rev. B

Editorial changes.



1.5 Typographic Conventions

Typographic Conventions can be found in the following document:

- Typographic Conventions



2 Overview

The CUDB system offers two types of consistency tools: the **Consistency Check** described in this document, and the **Lightweight Consistency Check** described in [CUDB Data Storage Handling](#).

Geographical data redundancy in CUDB is achieved by asynchronous data replication from the master replicas of each DSG or the PLDB to the corresponding slave replicas.

Changes on the data stored in the master replica typically take a short time to be replicated to the slave replicas. The time between the moment a piece of data is changed in the master replica, and the moment that change is applied on the slave replica is called replication lag.

The **CUDB Consistency Check** function is used to verify if there are any data differences between a slave replica and its corresponding master replica in a DSG or PLDB. During normal operation, data divergence cannot happen, but it can appear as a result of software bugs, or wrong operation and maintenance.

When a check task is ordered, it is queued for execution. Checks run on the node holding the PLDB or DSG master replica, and only one check can be running at the same time in a given CUDB node. The actual execution may happen after the previously ordered checks are completed, if there were any.

The CUDB Consistency Check function accesses the database servers of the replicas to check, compares the replicas, and then presents the result in output log files.

The check raises alarms in case data differences are found between the master and slave replicas. The alarm severity depends on the amount of differences found during the check, and if any schema difference is found.

The check tasks perform database queries over the network, write output files, thereby they consume system resources. To limit the amount of resources used simultaneously by the function, the check tasks are executed in a sequential manner according to the following logic:

- A task can be ordered from any node in the CUDB system. See [Ordering the Consistency Check](#) on page 6 for more information.
- After ordering, the task is placed into the Pending Task List (PTL) of the site where the current PLDB or DSG master is located, and marked to be executed on the PLDB or DSG master node.
- Each check task is picked from the PTL in a First-In First-Out (FIFO) manner, and moved to the Running Task List (RTL). After that, it is executed on the node where the PLDB or DSG master is running, unless the PLDB or DSG mastership has changed since ordering the task, in which case the task is discarded.



- Only one check task is executed on a node at a time. In case the node has multiple PLDB or DSG masters, the tasks are executed sequentially.
- The contents of the PTL and RTL can be listed for the whole CUDB system. See [Listing Tasks](#) on page 7 for more information.

2.1 Execution Conditions

When using CUDB Consistency Check, some conditions must be met to ensure that the check runs correctly, and that PLDB or DSG replica performance degradation is avoided. Therefore, make sure that the following conditions are met before using the feature:

- No LDAP schema change is performed while a Consistency Check task is running.
- No software installation, upgrade, or rollback is executed while a Consistency Check task is running.

Immediately before and during the execution of a Consistency Check task, the checked Data Store (DS) units of the PLDB or DSG are automatically monitored for degradation, to avoid overloading the PLDB or DSG replica. The results of such degradation are as follows:

- The check task does not start and the failure is logged if either of the following happens before starting the check:
 - DS unit degradation is detected (to avoid overload).
 - Replication goes down for the checked slave (to limit output size and avoid false positive results).
- An ongoing check task stops, its results are discarded, and the failure is logged if any of the following happens during its execution:
 - DS unit degradation is detected (to avoid overload).
 - Replication stops for the checked slave (to limit output size and avoid false positive results).
 - Mastership change occurs within the PLDB or DSG (to avoid generating invalid output).
- There are a few seconds when the execution of a Consistency Check task can fail to start if either of these happens:
 - A Consistency Check task is ordered immediately after a PLDB or DSG mastership change.
 - A mastership change occurs on the PLDB or DSG just before starting a previously ordered Consistency Check task.



Note: The reason for this limitation is that the replication lag information is not available in the system in the short period following the PLDB or DSG mastership change.

In case any kind of configuration change is performed on a CUDB node (for example, topology changes like adding DSGs, CUDB nodes, moving nodes between sites, changing network connection information), the previously ordered tasks that are in conflict with the changed configuration can be discarded from the PTL. This can be checked by listing the task history with the `cudbConsistencyMgr -t` command. See [Reviewing the Task History](#) on page 8 for more information.

In addition, if the administrative operation `applyConfig` is executed after a configuration change, the running consistency check tasks can be aborted if the change concerns configuration data used by the Consistency Check function.



3 Consistency Check

This section describes how the CUDB Consistency Check function works and how it can be used.

To find out how to execute commands on a CUDB node, refer to the CUDB CLI section of *CUDB System Administrator Guide*.

3.1 Ordering the Consistency Check

This section shows how to order the check for the replicas. To order a check task, the target slave replica must be fully operational (that is, not degraded), accessible (that is, replication must work), and the PLDB or DSG must have a working master.

Consistency check can be ordered both for DSG and PLDB replicas as described below.

- Use the following command to order a check task for a DSG replica on one of the System Controllers (SCs):

```
SC_2_1# cudbConsistencyMgr --order ms --node <nodeid> --dsg  
<dsgid>
```

In the above command, `<nodeid>` specifies the ID of the CUDB node to be checked, while `<dsgid>` specifies the DSG that is the subject of the Consistency Check.

- Use the following command to order a check task for a PLDB replica:

```
SC_2_1# cudbConsistencyMgr --order ms --node <nodeid> --pl
```

If the ordering is successful, then `cudbConsistencyMgr` prints the task ID, which can be used to track the task. Along with the task ID, the command also prints the ID of the CUDB node where the task is executed. In the example below, the node ID is 42:

Task UTC_2014-09-10-11-41-32_N42_U0000000035 is put into the pending task list and will be executed on node 42.

Optional Arguments

The following optional arguments are available for Consistency Check:

- **--max-replica-lag <milliseconds>** : This parameter sets the maximum tolerable replication distance between the master and the slave. The value is provided in milliseconds, and defines the execution conditions for the check. The default value is 10000: this means that the Consistency Check task is not



started (and respectively, a running task is stopped), if the replication lag reaches or exceeds 10000 ms.

- **--alarm-severity-limit <threshold>** : This parameter controls the severity of the alarm that is raised when inconsistency is found. If the number of divergences found is higher than the value set for <threshold> , then a major severity alarm is raised instead of a minor one. If the <threshold> value is not specified, then the default value of 500 is used.

3.2 Listing Tasks

The check tasks currently in the system can be listed by executing the following command:

```
SC_2_1# cudbConsistencyMgr -l
```

The output of the command uses two lists:

- Pending Task List (PTL) for ordered tasks.
- Running Task List (RTL) for currently running tasks.

See [Example 1](#) below for an example output:

Example 1 Task Listing of cudbConsistencyMgr -l

```
CUDB_121 SC_2_1# cudbConsistencyMgr -l
```

```
[Site 1]
```

```
RTL: UTC_2014-09-09-11-00-10_N121_U0000000849
      checkType=ms,source=S1-N121-D1,check=S2-N122-D1,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
PTL:
```

```
[Site 2]
```

```
RTL: UTC_2014-09-09-11-00-16_N122_U0000000793
      checkType=ms,source=S2-N122-D2,check=S1-N121-D2,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
PTL: UTC_2014-09-09-11-00-17_N122_U0000000794
      checkType=ms,source=S2-N122-D2,check=S1-N121-D2,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
      UTC_2014-09-09-11-00-21_N122_U0000000795
      checkType=ms,source=S2-N122-D3,check=S1-N121-D3,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
      UTC_2014-09-09-11-00-22_N122_U0000000796
      checkType=ms,source=S2-N122-D3,check=S1-N121-D3,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
      UTC_2014-09-09-11-00-23_N122_U0000000797
      checkType=ms,source=S2-N122-D3,check=S1-N121-D3,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
      UTC_2014-09-09-11-00-23_N122_U0000000798
      checkType=ms,source=S2-N122-D3,check=S1-N121-D3,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
      UTC_2014-09-09-11-00-23_N122_U0000000799
      checkType=ms,source=S2-N122-D3,check=S1-N121-D3,maxReplicaLag=10000,
      alarmSeverityLimit=500,verboseMode=off,debugMode=off
```

```
CUDB_121 SC_2_1#
```

The output consists of the following elements:



- `checkType`: The type of the consistency check. In the above example, `ms` stands for master - slave consistency check.
- `source=S<siteId>-N<nodeId>-D<dsgId>` : Identifies the master replica used for the check.
- `check=S<siteId>-N<nodeId>-D<dsgId>` : Identifies the slave replica to be checked.
- `maxReplicaLag=<integerValue>` : Shows the value of the `maxReplicaLag` attribute.
- `alarmSeverityLimit=<integerValue>` : Shows the value of the `alarmSeverityLimit` attribute.

Note: In case `D0` appears in the output, it refers to the PLDB.

3.3 Reviewing the Task History

The main points of task execution are recorded in the task history locally on the PLDB or DSG master node together with a timestamp and task ID.

Execute the following command to review the node-local history of check tasks:

```
SC_2_1# cudbConsistencyMgr -t
```

The output consists of log lines describing when each task was added to the list of pending tasks, when they started and stopped, and what was the result of the execution. The results are ordered by time, and are listed from the oldest to the newest.

```
2015-01-23 16:39:15+01:00 SC_2_2
UTC_2015-01-23-15-39-15_N119_U0000000000: task found in PTL:
checkType=ms,source=S1-N119-D66,check=S2-N120-
D66,maxReplicaLag=10000,alarmSeverityLimit=500,verboseMode=off,de
bugMode=off
```

```
2015-01-23 16:39:15+01:00 SC_2_2
UTC_2015-01-23-15-39-15_N119_U0000000000: task execution started
```

```
2015-01-23 16:44:00+01:00 SC_2_2
UTC_2015-01-23-15-39-15_N119_U0000000000: task execution finished
with result: Successful completion, no difference found. Exit
code: 0.
```

The possible exit codes and their meanings are as follows:

- Successful completion, no difference found. Exit code: 0
- Successful completion, differences found. Exit code: 1
- Termination due to error (for example server timeout, and so on). Exit code: 2



- Successful completion, database checked only partially because the Potential Problem List limit was exceeded. Exit code: 3

Exit codes 0 and 1 mean that the PLDB or DSG database has been scanned to the end, and valid output is written.

Exit code 2 means that the output of the check task has been discarded.

Exit code 3 means that only a portion of the PLDB or DSG database has been scanned, and the output written covers the scanned part.

The list is saved in size limited log files containing about at least two weeks of data.

3.4 Removing Tasks

Execute the following command to try removing an ordered task from the pending task list:

```
SC_2_1# cudbConsistencyMgr --remove <taskId>
```

With the above command, the system attempts to remove the task from the list of pending tasks only. It does not remove the task if it is already started and running. Use the `--force` option to try removing a task from the running task list:

```
SC_2_1# cudbConsistencyMgr --remove <taskId> --force
```

In the above commands, `<taskId>` is the task ID as it appears in the output of `cudbConsistencyMgr -l`.

The output of the above commands must be similar to the below example:

```
CUDB_166 SC_2_1# cudbConsistencyMgr --remove UTC_2014-09-22-07-20-04_N166 →
_U0000000011
--force
Task: UTC_2014-09-22-07-20-04_N166_U0000000011
successfully removed from RTL
```

Note: Even if the `--force` option is used, the task may finish before it can be removed.

A consistency check task can be started only if the source and target DS of the checked PLDB or DSG is alive (as displayed by the `cudbSystemStatus` command), and if they have not changed since the ordering. If a DS state changes from alive to something else (such as degraded, stopped, maintenance), or if the PLDB or DSG mastership changes, then any running task associated with the PLDB, DSG, or the faulty DS is stopped and removed from the RTL. If an ordered task is associated with a not alive DS, PLDB, or with a DSG that had mastership change since the ordering, then the ordered task is not moved to the RTL, and is removed from the PTL.



For the complete list of `cudbConsistencyMgr` options, execute `cudbConsistencyMgr --help`, or refer to [CUDB Node Commands and Parameters](#) for more information.

3.5 Consistency Check Output

The log files produced by CUDB Consistency Check are stored on the disk storage system of the SC where the check task was executed, in the following directory:

```
/local/cudb_ddci/replica_check
```

Note: The task executor CUDB node is the node that hosts the master replica of the checked PLDB or DSG, not the CUDB node on which the `cudbConsistencyMgr` command was executed. At the same time, the task executor SC is the SC that hosted the HA active DDCI Manager instance at the time when the task execution started. Use the following command to determine the SC on which the DDCI Manager is HA active:

```
SC_2_1# cudbHaState | grep DDCI
```

The expected output must be similar to the below example:

```
saAmfSISUHASState. "safSu=SC-1,safSg=2N,safApp=ERIC-CUDB_CUDBDDCI". ->
"safSi=2N-1":
active(1)
saAmfSISUHASState. "safSu=SC-2,safSg=2N,safApp=ERIC-CUDB_CUDBDDCI". ->
"safSi=2N-1":
standby(2)
```

Each Consistency Check task writes three files to the directory mentioned above:

- The LDAP tree log with the following file name: `cudbDsuDiff_tree_<TASK-ID>_S<FROM-SITE-ID>_N<FROM-NODE-ID>_S<TO-SITE-ID>_N<TO-NODE-ID>_DSG<DSG-ID>.xml`
- The table log with the following file name: `cudbDsuDiff_table_<TASK-ID>_S<FROM-SITE-ID>_N<FROM-NODE-ID>_S<TO-SITE-ID>_N<TO-NODE-ID>_DSG<DSG-ID>.log`
- The execution log with the following file name: `cudbDsuDiff_exec_<TASK-ID>_S<FROM-SITE-ID>_N<FROM-NODE-ID>_S<TO-SITE-ID>_N<TO-NODE-ID>_DSG<DSG-ID>.log`

The table log and the execution log contain internal CUDB information, and are intended for Ericsson support personnel only.

If no differences are found between the two replicas, or the found inconsistencies are identified only on the LDAP entry DN level, then the table log file is removed, as it would be empty.



The above behavior is affected by the verbose mode as follows:

3.5.1 LDAP Tree Log

The LDAP tree log contains information about the inconsistencies found by the Consistency Check. The log is in XML format, and is located in the log directory mentioned in [Consistency Check Output](#) on page 10.

Each Consistency Check task writes its own separate LDAP tree log file.

Attention!

The LDAP tree log records master-slave differences of the LDAP Directory Information Tree (DIT) tree, and contains LDAP object Distinguished Names (DN), object classes, and attribute values where applicable. Pieces of sensitive information about subscribers or system operations can be extracted from it by reverse-engineering and data mining techniques. To prevent such incidents, the access permissions of the LDAP tree log file are set to the same security level as the CUDB subscriber database itself.

3.5.1.1 Format of the LDAP Tree Log XML

[Example 2](#) shows an example XML of the LDAP tree log.

Example 2 LDAP Tree Log

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- LDAP tree log file for the CUDB replica consistency check -->
<CudbConsistencyAudit fromNode="N<from_node_id>" toNode="N<to_node_id>" dsg="<dsg_id>" execBlade="SC →
_2_[1|2]"
params="<cmdline_params>">
  <!-- If the problem is identifiable on LDAP entry level: group all object classes of a DN under a →
  //
  single LdapError tag -->
    <LdapError dn="<dn>">
      <DsuFrom> <!-- empty tag if the entry doesn't exist on this DSU -->
        <!-- otherwise list the differing object classes and attributes.
        If an object class is missing on a DS unit, it is only listed without its attributes unde →
        r the //
        tag of the DS
        unit where it is present -->
        <ObjectClass name="<obj_class>">
          <Attribute name="<attrib_name>" [value="<value>"] <!-- missing if difference only in MV attr →
          ibute //
          set or this is a BLOB -->
          [potential="yes"] <!-- present if difference is only potenti →
          al //
          (BLOB attributes) --> />
          <!-- more attributes -->
          </ObjectClass>
        </DsuFrom>
        <DsuTo> <!-- empty tag if the entry doesn't exist on this DSU -->
          <!-- otherwise list the differing attributes -->
          <ObjectClass name="<obj_class>">
            <Attribute name="<attrib_name>" [value="<value>"] [potential="yes"]/>
            <!-- more attributes -->
            </ObjectClass>
          </DsuTo>
        </LdapError>
      <!-- more LDAP errors -->

    <!-- If there is one problem at least, which is identifiable only on DB level, indicate as //
```



```
Internal Error: -->
  <InternalErrorsPresent />
</CudbConsistencyAudit>
```

If a Consistency Check task does not find any inconsistencies, then the resulting LDAP tree log file contains only the empty root `<CudbConsistencyAudit>` XML element.

If an inconsistency in the database can be associated with an LDAP object, then the corresponding entry in the LDAP tree log file is indicated with an `<LdapError>` XML element, which contains the full DN of the LDAP object in its `<dn>` XML attribute. Depending on the semantic meaning of the inconsistency, additional LDAP-level information can be provided: if the inconsistency can be narrowed down to a specific LDAP object class, or even to a specific LDAP attribute of the LDAP object, then they are indicated with XML sub-elements (`<ObjectClass>` and `<Attribute>` respectively) under the corresponding `<LdapError>` XML element. This additional information is recorded and presented from both the master and the slave PLDB or DSG replicas (in the `<DsuFrom>` and `<DsuTo>` XML elements) to help troubleshooting.

Note: More than one inconsistency can occur for the same LDAP object. Even if this happens, only one `<LdapError>` XML element is created for the LDAP object in the LDAP tree log, and the additional troubleshooting information of the separate inconsistencies is merged together under this single XML element.

The consistency checking process also includes the Binary Large Object (BLOB)-type LDAP attributes. However, for performance reasons, these are omitted from the final checking phase, during which the inconsistent items are transferred to the SC that hosts the running Consistency Check task. If an LDAP object has inconsistent LDAP attributes belonging to such an LDAP object class that has BLOB-type attributes as well, then all BLOB-type attributes of that LDAP object class are also listed as potential inconsistencies, marked with a `potential=yes` XML attribute in the LDAP tree log. The values of the BLOB-type attributes are not indicated in the LDAP tree log.

If there is an inconsistency in a binary-type LDAP attribute (that is, the value is not textual data), then the raw LDAP attribute value is presented as a byte sequence, where each byte value is transcoded to its hexadecimal representation. The byte-sequence is prepended by `hex:` in this case. Bit-type attribute values are presented as a bit sequence with the `bin:` prefix.

The output of the differing binary-type and textual-type LDAP attribute values is limited to the first 255 characters of their textual representation. If the length of the value string is longer than this limit, it is truncated to the limit and a `"..."` suffix is appended to the value.

If an inconsistency cannot be associated with an LDAP object, then it is indicated in the LDAP tree log with an `<InternalErrorsPresent>` element.



Note: For performance reasons, the identification of LDAP DN inconsistencies is performed based on the database contents of the (locally available) master PLDB or DSG replica only. If the required information is present only in the (remote) slave PLDB or DSG replica, then the DN identification is skipped, and such inconsistencies are reported with the `<InternalErrorsPresent>` XML element.

In case of an LDAP schema inconsistency, no `<LdapError>` tags are used, only an `<InternalErrorsPresent>` element.

DN identification is not possible in the following cases:

3.5.1.2 Structure of the LDAP Tree Log XML

The LDAP tree log XML file consists of the following parts:

- Log header:

```
<?xml version="1.0" encoding="UTF-8"?>

<CudbConsistencyAudit fromNode="<from_node_id>"
toNode="<to_node_id>" dsG="<dsG_id>" execBlade="<SC_2_[1|2]>"
params="<cmdline_params>">
```

- List of the problems identified on LDAP entry DN level. The problems are listed in the `<LdapError>` elements. The differing attribute of the LDAP object belongs to the specific reported object class:

```
<LdapError dn="<identified LDAP entry DN level of the problem>" >
<DsuFrom>
<ObjectClass name="<object class name>">
<Attribute name="<attribute name>" [value="<attribute value>"] [potential=
"yes"] />
</ObjectClass>
</DsuFrom>
<ObjectClass name="<object class name>">
<Attribute name="<attribute>" [value="<attribute value>"] [potential="yes →
"] />
</ObjectClass>
</DsuTo>
</LdapError>
<!-- ... (Further LdapError tags, if any. The order of LdapError elem→
ents is unspecified.) ... -->
```

- In case there is a problem (or problems) that cannot be identified on LDAP entry DN level, it is indicated by an `<InternalErrorsPresent>` tag:

```
<InternalErrorsPresent />
```

- Log footer:



```
</CudbConsistencyAudit>
```

3.5.1.3 LDAP Tree Log Output Examples

This section contains some LDAP tree log output examples, depicting the common output scenarios.

Example 3 LDAP Tree Log - No Differences

[Example 3](#) shows how the tree log XML output looks like in case there are no differences between the replicas:

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
--to-node 165 --dsg 0 //
--id UTC_2014-09-04-15-20-51_N166_U00000000000 --max-replica-lag 10000 --alarm-se →
verity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
</CudbConsistencyAudit>
```

Example 4 LDAP Schema Sanity Check Problem

[Example 4](#) shows an LDAP tree log that indicates an LDAP schema sanity check problem. The `<InternalErrorsPresent>` element shows this:

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
--to-node 165 --dsg 0 //
--id UTC_2014-09-04-15-20-51_N166_U00000000000 --max-replica-lag 10000 --alarm-se →
verity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
<InternalErrorsPresent />
</CudbConsistencyAudit>
```

Any problem that cannot be identified on LDAP level is indicated by an `<InternalErrorsPresent>` element.

Example 5 LDAP Tree Log - Difference in a Single-Value LDAP Attribute

[Example 5](#) shows an LDAP tree log containing only one problem. However, this problem was identified on LDAP level, so it appears as an `<LdapError>` element with the full DN of the problematic LDAP object. The values exist both on the master and the slave PLDB replicas, so they can be presented in the LDAP tree log.

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
```



```
--to-node 165 --dsg 0 //
--id UTC_2014-09-04-15-20-51_N166_U0000000000 --max-replica-lag 10000 --alarm-severity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
<LdapError dn="serv=csp,mscId=856,ou=multiSCs,ou=ft,o=cudb,c=es">
<DsuFrom>
<ObjectClass name="CsPsMSISDNSubscriber">
<Attribute name="NAM" value="1" />
</ObjectClass>
</DsuFrom>
<DsuTo>
<ObjectClass name="CsPsMSISDNSubscriber">
<Attribute name="NAM" value="2" />
</ObjectClass>
</DsuTo>
</LdapError>
</CudbConsistencyAudit>
```

Example 6 LDAP Tree Log - Difference in a Multi-Value LDAP Attribute

Unlike the previous example, [Example 6](#) contains a difference related to a multi-value attribute. Although the values exist both on the master and the slave PLDB replicas, they are not presented in the LDAP tree log to avoid confusion, as there can be more than one value stored in this type of LDAP attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N165" toNode="N166" dsg="0" execBlade="SC_2_1" params="-f 165 -t 166 -d 0 -o output_dir // --id id">
<LdapError dn="ou=associations,ou=ft,o=cudb,c=es">
<DsuFrom>
<ObjectClass name="organizationalUnit">
<Attribute name="ou" />
</ObjectClass>
</DsuFrom>
<DsuTo>
<ObjectClass name="organizationalUnit">
<Attribute name="ou" />
</ObjectClass>
</DsuTo>
</LdapError>
</CudbConsistencyAudit>
```

Example 7 LDAP Tree Log - Missing Object Class or Multi-Value Attribute on the Slave Replica

It can happen that data are missing from the slave PLDB or DSG replica (that is, an LDAP object has a missing object class, or a missing multi-value attribute value in an object class), but the LDAP DN of the object can be identified. In such cases, as shown in [Example 7](#), only the problematic object class is reported, as there is no information on the slave PLDB or DSG replica for value comparison.



```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
--to-node 165 --dsg 0 //
--id UTC_2014-09-04-15-20-51_N166_U00000000000 --max-replica-lag 10000 --alarm-se →
verity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
<LdapError dn="ou=associations,ou=ft,o=cudb,c=es">
<DsuFrom>
<ObjectClass name="organizationalUnit">
</ObjectClass>
</DsuFrom>
<DsuTo>
</DsuTo>
</LdapError>
</CudbConsistencyAudit>
```

Example 8 LDAP Tree Log - Multiple Differences in the Same LDAP Object

[Example 8](#) shows a scenario where there are two problems in the same `<LdapError>` tag. This is because the identified LDAP object is the same for both problems.

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
--to-node 165 --dsg 1 //
--id UTC_2014-09-04-15-20-51_N166_U00000000000 --max-replica-lag 10000 --alarm-se →
verity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
<LdapError dn="serv=csp,mscId=8,ou=multiSCs,ou=ft,o=cudb,c=es">
<DsuFrom>
<ObjectClass name="CsPsOptFeatData">
<Attribute name="ACC" value="2" />
</ObjectClass>
<ObjectClass name="CsPsSubscVersData">
<Attribute name="CSP" value="131" />
</ObjectClass>
</DsuFrom>
<DsuTo>
<ObjectClass name="CsPsOptFeatData">
<Attribute name="ACC" value="1" />
</ObjectClass>
<ObjectClass name="CsPsSubscVersData">
<Attribute name="CSP" value="130" />
</ObjectClass>
</DsuTo>
</LdapError>
</CudbConsistencyAudit>
```



Example 9 LDAP Tree Log - Potential Difference in a BLOB-type LDAP Attribute

Example 9 shows a log indicating a potential difference in a BLOB-type LDAP attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<CudbConsistencyAudit fromNode="N166" toNode="N165" dsg="0" execBlade="SC_2_2" p →
arams="--from-node 166
--to-node 165 --dsg 1 //
--id UTC_2014-09-04-15-20-51_N166_U0000000000 --max-replica-lag 10000 --alarm-se →
verity-limit 3 //
--output-dir /local/cudb_ddci/replica_check">
<LdapError dn="ImsCxDynInfId=ImsCxDynInf,ImsSubsId=ImsSubs,serv=IMS,assocId=0011 →
300033,ou=associations,dc=operator,dc=com">
<DsuFrom>
<ObjectClass name="ImsCxDynInf">
<Attribute name="ImsSubsData" potential="yes" />
</ObjectClass>
</DsuFrom>
<DsuTo>
<ObjectClass name="ImsCxDynInf">
<Attribute name="ImsSubsData" potential="yes" />
</ObjectClass>
</DsuTo>
</LdapError>
</CudbConsistencyAudit>
```

3.5.2 Log Cleanup

While a check is running, the names of the table log and the LDAP tree log files have a .part suffix, indicating that the file is not yet complete (they are partial). When the check completes, the .part suffix is stripped from the file names. However, if the check is aborted or terminates unexpectedly, these incomplete files are deleted by the high-availability manager process. In addition, an output file cleanup is performed at the start of the manager process to remove any leftover file that ends with .part.

When a check completes, or the manager process initiates, the manager process removes log files that are older than two weeks to avoid using up the disk storage system space. In addition, if the combined size of all log files is larger than 300 MB, then additional log files are deleted, starting from the oldest ones. The log files of the latest successfully completed task are always kept.

Note: Due to the potential High Availability (HA) state changes of the manager process and PLDB or DSG mastership changes, check the output directory on each SC and each node.



3.6 Alarms

The Consistency Check tool raises alarms on the node that holds the slave replica, if inconsistencies are found during the check. Four alarms can be raised; the alarm selection depends on the following conditions:

- Where the check is performed (the PLDB or a DSG).
- The number of inconsistencies found on internal database level.
- Whether the LDAP Schema Sanity Check (part of the Consistency Check) finds any schema inconsistency between the two replicas.

The available alarms are as follows:

- Storage Engine, Data Inconsistency between Replicas Found in DS, Minor
- Storage Engine, Data Inconsistency between Replicas Found in DS, Major
- Storage Engine, Data Inconsistency between Replicas Found in PLDB, Minor
- Storage Engine, Data Inconsistency between Replicas Found in PLDB, Major

Avoiding Duplicate Alarms

The Consistency Check makes sure that no outdated alarms remain in the system. Therefore, if the Consistency Check is successful (that is, new information is available on the inconsistencies in the database), it clears the alarm raised by a previous Consistency Check for the same DSU pair before checking if a new alarm must be raised. This means the following:

- If the current Consistency Check is run in the PLDB, it checks for minor and major alarms in the PLDB indicating inconsistency between the same replicas that the current check was checking. The tool then clears them, if it finds any.
- If the current Consistency Check is run in a DSG, it checks for minor and major alarms in the same DSG and replica that the current Consistency Check was checking. The tool then clears them, if it finds any.

Note: The tool can raise multiple alarms with the same name. However, similar alarms cannot be raised with the same parameters, meaning that alarms with the same name are not caused by the same reasons (for example, if multiple DS alarms are raised with the same name, they are probably raised for different DS units). Such alarms are of course not cleared until a relevant Consistency Check is executed, or are cleared manually.

Raising and Clearing Alarms

The logic of raising and clearing replica consistency alarms are as follows:



- *Raising alarms:* The Consistency Check raises alarms if it finds any inconsistency, and does not raise any, if it does not. The type of alarm raised (that is, major or minor) depends on the number of inconsistencies found, if any schema difference is found, and the value of the `--alarm-severity-limit` parameter. The Consistency Check raises a minor alarm if the number of inconsistencies found does not exceed the value of the parameter and no schema difference is found, and raises a major alarm otherwise.

If the Consistency Check is executed in the PLDB, the raised alarm contains the IDs of the compared nodes, and the task ID. If it is executed in a DSG, it contains the DSG ID, and the node ID of the slave replica.

- *Clearing alarms:* Each alarm can be cleared manually, or is cleared at the next execution of the Consistency Check on the same PLDB or DSG replicas, if no differences are found, or if the alarm is replaced with another data inconsistency alarm. Refer to [CUDB Node Fault Management Configuration Guide](#) for more information on how to clear alarms manually.

3.7 Use Recommendations

When using the CUDB consistency check tools, consider the following recommendations:

- The CUDB system offers two types of consistency check tools:
 - The Lightweight Consistency Check, performed by the `cudbCheckConsistency` command. This command performs a quick check by comparing only the number of rows in the database tables of the PLDB or DSG master and slave replicas.
 - The Consistency Check, which compares the contents of the database tables containing LDAP entry attribute values, and performs a deep check. CUDB Consistency Check is recommended to be used regularly to monitor and keep the consistency of the data between the PLDB or DSG replicas.
- The preferred way of checking is to compare each PLDB or DSG slave replica against the master replica of the PLDB, or the specific DSG. These checks must be executed sequentially, as they are all executed on the PLDB or DSG master node. Still, all the checks can be ordered for execution. In this case, the check tasks are queued in the Pending Task List (PTL).
- The execution time of the check is proportional to the amount of data stored in the PLDB or DSG, and is specific to the PLDB or that DSG. The execution time can be calculated as follows:
 1. Order a check for the concerned PLDB or DSG slave.
 2. Wait for its completion.
 3. Issue the following command:

**cudbConsistencyMgr -t**

When executed, check for check tasks whose exit code is 0 or 1. These values mean that the entire database content was scanned. See [Example 10](#) for an example.

The execution time can be calculated from the `task execution started` and `task execution finished` lines. See [Reviewing the Task History](#) on page 8 on how to review the task history.

- The main goal of the consistency tools is to keep the database replicas identical for data consistency. In case differences are found, the recommended procedure described in the alarm OPIs (see [Alarms](#) on page 18) is to perform recovery as soon as the alarm is raised. Two scenarios can occur:
 - Small amount of differences are detected, and no schema differences are found (a minor alarm is raised). In this case, recovery or reprovisioning can be attempted manually, based on the LDAP DN's found in the LDAP tree log.
- Note:** The reprovisioning procedure is outside the scope of this document.
- Large amount of differences are detected, or schema differences are found (a major alarm is raised). Manual reprovisioning or recovery is not feasible in this case. The preferred way to make the replicas identical is a combined unit data backup and restore on the affected PLDB or DSG. For further information about the combined unit data backup and restore procedure, refer to [CUDB Backup and Restore Procedures](#).

The decision on whether the differences are minor or major is based on the `--alarm-severity-limit` parameter of the check specified to the `cudbConsistencyMgr` command. The limit is measured in database table rows, that mostly correspond 1-to-1 to LDAP attribute value differences.

If recovery is done, it is recommended to run the check on the other replicas of the PLDB or the same DSG. Before performing the check, check the PTL with the following command to see if any of the wanted checks are already ordered:

```
cudbConsistencyMgr --list
```

Example 10 cudbConsistencyMgr -t Output Example

```
# cudbConsistencyMgr -t | grep UTC_2014-10-08-11-50-41_N119_U0000000178
2014-10-08 13:50:41+02:00 SC_2_1 UTC_2014-10-08-11-50-41_N119_U0000000178: task found in PTL: checkT →
ype=ms, //
source=S1-N119-D1 check=S2-N120-D1,maxReplicaLag=10000,alarmSeverityLimit=100
2014-10-08 13:50:41+02:00 SC_2_1 UTC_2014-10-08-11-50-41_N119_U0000000178: task execution started
2014-10-08 13:55:58+02:00 SC_2_1 UTC_2014-10-08-11-50-41_N119_U0000000178: task execution finished w →
ith //
result: Successful completion, differences found. Exit code: 1.
```



Glossary

For the terms, definitions, acronyms and abbreviations used in this document, refer to CUDB Glossary of Terms and Acronyms



Reference List

CUDB Documents

1. CUDB Data Storage Handling
2. CUDB Node Commands and Parameters
3. CUDB System Administrator Guide
4. Storage Engine, Data Inconsistency between Replicas Found in DS, Minor
5. Storage Engine, Data Inconsistency between Replicas Found in DS, Major
6. Storage Engine, Data Inconsistency between Replicas Found in PLDB, Minor
7. Storage Engine, Data Inconsistency between Replicas Found in PLDB, Major
8. CUDB Node Fault Management Configuration Guide
9. CUDB Backup and Restore Procedures
10. CUDB Glossary of Terms and Acronyms