

CUDB LDAP Interwork Description

INTERWORK DESCRIPTION

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	Scope	1
1.2	Revision Information	1
1.3	Target Groups	3
1.4	Typographic Conventions	3
2	Overview	5
3	CUDB Data Model Description	7
3.1	CUDB Main Directory Information Tree	7
3.2	LDAP Views	19
4	Attributes and Object Classes	21
4.1	Attributes	21
4.2	Object Classes	22
4.3	Operational Attributes	24
4.4	Limitations	24
5	CUDB DIT Entries	27
5.1	CUDB Admin Entry	27
5.2	Identities Container Entry	27
5.3	Multi Service Consumer Container Entry	27
5.4	Associations Container Entry	28
5.5	Multi Service Consumer Common Data Container Entry	29
5.6	Service Common Data Container Entry	30
5.7	dsgLocks Container Entry	30
6	Operations	33
6.1	CDC	35
6.2	Base DN Transformation in LDAP operations over Authentication Data Operations (serv=Auth)	37
6.3	Performing LDAP Search Below the DE	38
6.4	Error Codes	38
6.5	LDAP Controls	41
6.6	CUDB LDAP Extended Operations	42



Glossary	45
Reference List	47



1 Introduction

This document describes the Ericsson reference data model for the basic and initial data maintained in the Ericsson Centralized User Database (CUDB), providing a detailed description of Lightweight Directory Access Protocol (LDAP) data model information, such as CUDB directory entries, object and attribute descriptions, formats, and allowed value ranges.

This document also provides an overview of supported LDAP operations and LDAP users that can be used to manage, provision and access the data stored in CUDB.

1.1 Scope

This document covers the following:

- CUDB data model description
- Attributes and objects classes
- CUDB DIT entries
- Operations to handle CUDB data

This document provides the necessary information and understanding of the basic structures of CUDB data model and how to use the LDAP interface to manage stored data to facilitate the integration of application Front Ends (FEs) data into CUDB.

The description of specific application FE data models or internal-only CUDB data is out of the scope of this document.

1.2 Revision Information

Rev. A

This document is based on 1/15519-HDA 104 03/9 with the following changes:

- Section 3.1.2 on page 9, Section 3.2 on page 19, and Section 6 on page 33: Updated information on function availability.
- Section 3.1.4 on page 12: Added information about container entries.
- Section 6.4 on page 38: Added error code 3 to Table 20 and updated proxy timeout information for error code 11.



Rev. B

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

- Section 6.5 on page 41: New section.

Rev. C

Editorial changes only.

Rev. D

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

- Section 6.4 on page 38: Updated error code 3, 11, 52 and 80 in Table 20.
- Section 6.5.1 on page 41: Updated ReadMode control value type.

Rev. E

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

- Section 3.1.7 on page 16: Updated LDAP search operation example and added a note.

Rev. F

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

- Section 6.1 on page 35: New chapter. The content has been moved from Section 4.1 on page 21.
- Section 6.4 on page 38: Updated table headers and error code 80 in Table 20.

Rev. G

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

- Section 5.4.2 on page 29: Updated the value in the Entry Name column of Table 11.
- Section 6.4 on page 38: Updated CUDb Error Description and Possible Client Actions of error 51 in Table 20.

Rev. H

- Section 3.1 on page 7: Added information related to new distribution entry `dsgLocks`. Updated Figure 1.



- Section 3.1.8 on page 18: New section.
- Section 4.1 on page 21: Section 4.1 Attributes: Updated Table 1 Attributes with new attributes related to `dsgLocks`.
- Section 4.2 on page 22: Section 4.2 Object Classes: Updated Table 2 with new object classes related to `dsgLocks`.
- Section 5.7 on page 30: New section.
- Section 5.7.1 on page 31: New section.
- Section 5.7.2 on page 31: New section.

Rev. J

- Section 4.1 on page 21: Updated the Attribute Object Identities of `cudbLdif` and `cudbLdifprov` attributes in Table 1.
- Section 4.4 on page 24: Updated the list of limitations on attributes and object classes used in CUDB.
- Section 6 on page 33: Added list of LDAP extended operations and Table 19.
- Section 6.3 on page 37: Added description of Optimized Subtree Searches function.
- Section 6.6 on page 42: Added section.

1.3 Target Groups

This document is intended for users working with CUDB LDAP schemas. This document assumes the general knowledge of the LDAP standard.

1.4 Typographic Conventions

Typographic conventions can be found in the following document:

- *Typographic Conventions*





2 Overview

CUDB is the network entity in a layered architecture providing a central data storage point for application FE data (Home Location Register (HLR) FE data, IP Multimedia Subsystem (IMS) data, and so on). Stored data is exposed and accessed through LDAP v3 protocol.

LDAP is a client-server based directory access protocol that provides both read and update access. The LDAP data information model provides the structures and data types necessary for building an LDAP directory tree.

This document describes the LDAP data provided in the following LDAP schemas:

- `/cluster/home/cudb/dataAccess/ldapAccess/ldapFe/config/schema/cudb.schema`
- `/cluster/home/cudb/dataAccess/ldapAccess/ldapFe/config/schema/application_counters.schema`





3 CUDB Data Model Description

The following sections describe CUDB data model. For more information on LDAP data model, refer to *CUDB LDAP Data Access*, Reference [1].

3.1 CUDB Main Directory Information Tree

In CUDB, from the LDAP data modeling perspective, the upper levels of the Directory Information Tree (DIT) are divided into the following branches:

- CUDB root entry
- Administrative LDAP user data (out of the scope of this document)
- Application Identity data.
- Multi Service Consumer (MSC) data.
- Association data.
- MSC common data.
- Service common data.
- Application Counters data (read only)
- `dsgLocks` data

Figure 1 shows CUDB LDAP DIT.

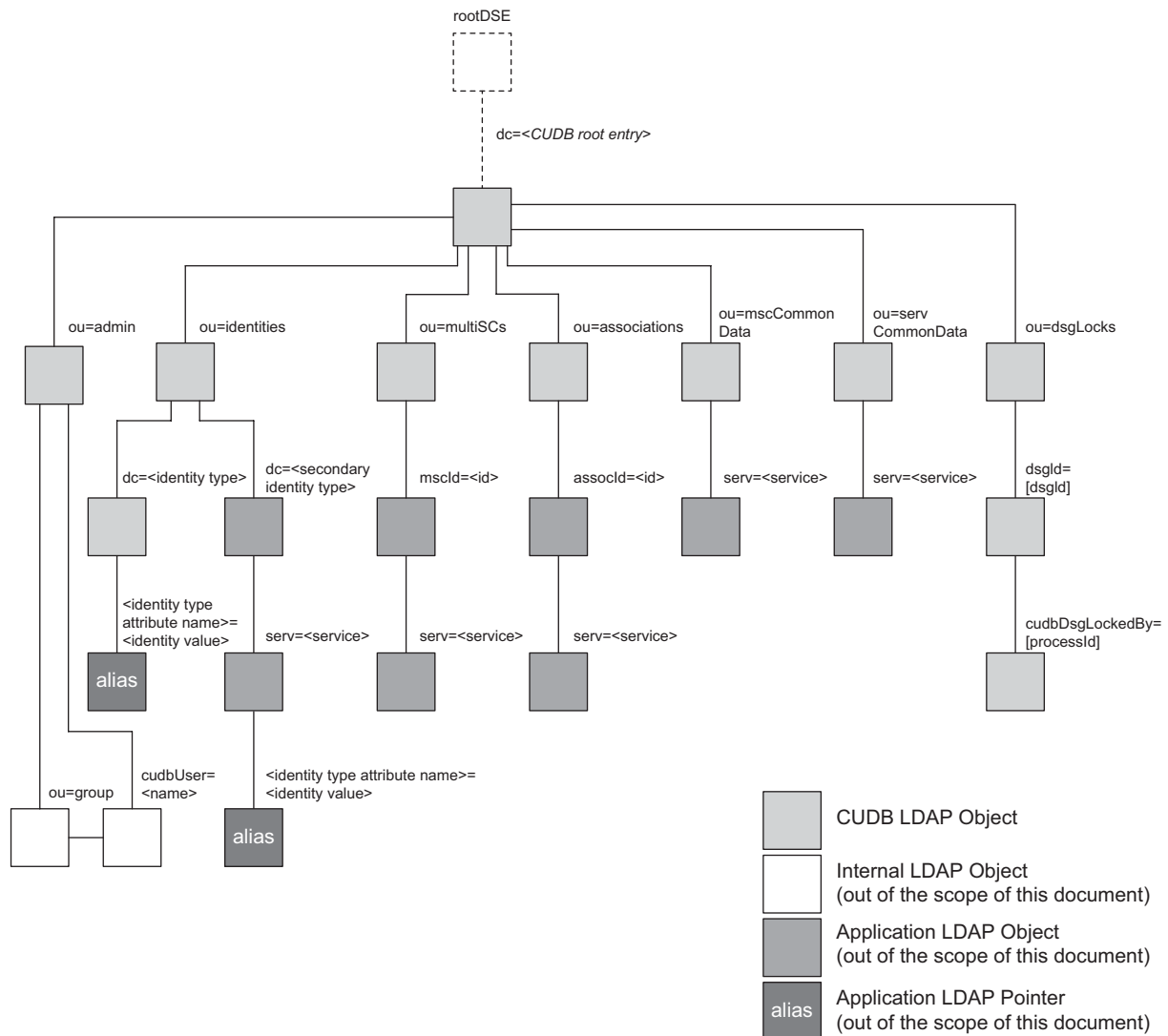


Figure 1 CUDB LDAP DIT

CUDB Data is distributed across data partitions. To achieve data distribution, CUDB provides a special type of entry, called a Distribution Entry (DE). CUDB DE are regular LDAP entries, but contains additional metadata to store and find the entries and their sub-trees in a given data partition

CUDB, by default, has three built-in DEs, but allows defining custom DEs, if necessary. The build in DEs are MSCs, Associations, and CUDBLockDistribution.

Custom DEs must fulfill the following requirements:

- When a DE is provisioned, the following two optional attributes must be defined in any included object class that is part of the DE:
 - ZoneId.



- DSUnitGroup.

For more information, refer to *CUDB Node Configuration Data Model Description*, Reference [3].

3.1.1 CUDB Root Entry

The CUDB root entry is the top level entry of the directory tree.

The Distinguished Name (DN) and Relative Distinguished Name (RDN) of the CUDB root entry are fully configurable at CUDB installation time.

3.1.2 CUDB LDAP Users and Groups

For an application FE to access the data stored in CUDB through LDAP interfaces, authentication must be completed before a session is established. Therefore, applications must create or use one or more LDAP users with the corresponding credentials and configuration parameters.

Once the LDAP user has been created, application FEs can bind to the LDAP interface and send LDAP operations to CUDB. Optionally, user groups can be defined to share access control rules. For more information on how LDAP users and LDAP users groups are defined in CUDB, refer to *CUDB System Administrator Guide*, Reference [5].

LDAP users have additional configuration parameters. Using the LDAP Data Views function, LDAP users are assigned an LDAP data view to access the data through a custom DIT defined for the view. Users without an assigned LDAP data view access the data through the default DIT.

Note: The LDAP Data Views function requires the Application Facilitator Value Package.

LDAP users and user groups are defined through configuration. For more information, refer to *CUDB Node Configuration Data Model Description*, Reference [3].

The administrative data entries defined in CUDB are the following:

- Admin container: "ou=admin,<CUDB root entry>"
- LDAP User data: "cudbUser=<CUDB LDAP user>,ou=admin,<CUDB root entry>"
- LDAP Group data: "ou=<CUDB LDAP group>,ou=admin,<CUDBroot entry>"
- LDAP User data belonging to a LDAP Group: "cudbUser=<CUDB LDAP user>,ou=<CUDB LDAP group>,ou=admin,<CUDBroot entry>"



3.1.3 Identity Data

Identities are LDAP entries containing a reference to find and access subscriber data, typically MSC or Association, data. Identity entries are placed under `ou="identities"` container level entry below the CUDB root entry.

There are two types of identity entries, depending on how they reference an MSC or an Association DE.

- **Primary identities:** These entries are always LDAP aliases referencing another entry (typically to an MSC, an Association, or to a secondary identity). As primary identities serve this specific purpose, they are special entries in CUDB, as each type must be defined at installation and the entries are stored to optimize the space taken up by each entry. Primary identities can only be stored directly below its type container, which is populated at installation time. Even though primary identities are LDAP aliases, CUDB supports a non-standard LDAP modify operation, that allows modifying the target entry through alias de-referencing, if the entry to be modified is addresses through a primary identity. For more details, see Section 6 on page 33.
- **Secondary identities:** These entries are similar to primary identities, as they are usually used to references other entries, but the alias does not have to be in the identity itself, but can also be below it. Unlike primary identities, these are not special entries in CUDB. For example, a secondary identity could be an entry that contains subentries for different services where each subentry contains an alias referencing to a different MSC or Association.

Figure 2 provides an example of primary and secondary identity entries.

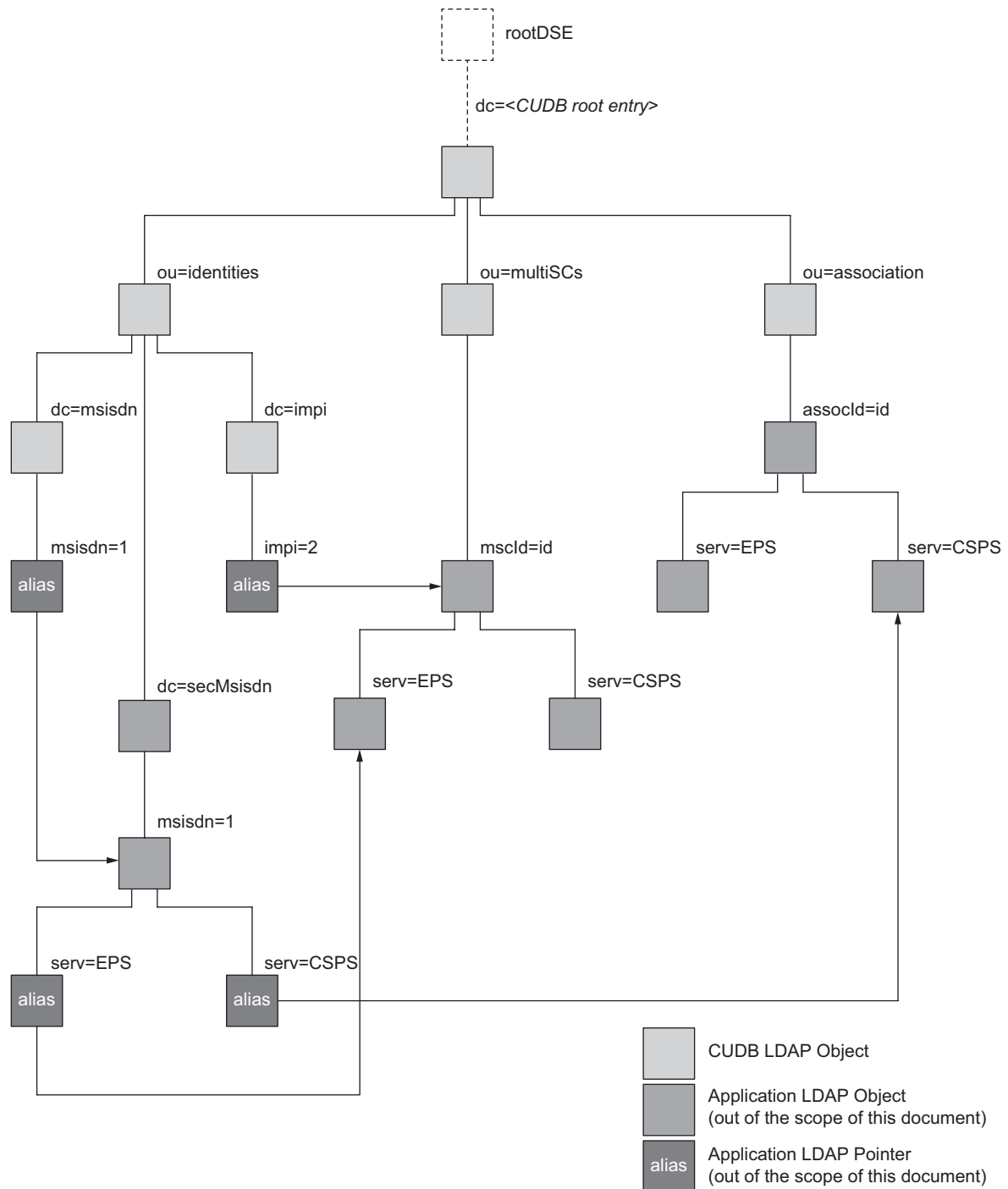


Figure 2 Primary and Secondary Identity Entries

A primary identity data entry, defined in CUDB:

- Identity container: "ou = identities, <CUDBroot entry>"



Separate containers for different identity types of identities can be defined based on identity data:

- Identity domain container: "dc=<identity type>, ou= identities, <CUDB root entry>"

<identity type> can take any value. For example, msisdn, imsi, impi, impu, ipaddress, and so on.

The entry must contain at least one LDAP object class containing the attribute dc. For example, dcObject.

Following is an example of container for MSISDN:

```
"dc = msisdn, ou = identities, <CUDB root entry>"
```

Under each different identity type container, alias entries per identity can be defined:

- Identity alias: "<identity type attribute name>=<identity value>, dc=<identity type>, ou = identities, <CUDB root entry>"

<identity type attribute name> can take any name. For example, msisdn, imsi, impi, impu, ipaddress, and so on.

This attribute, its type, syntax, and LDAP object class are defined and provided in an LDAP schema by CUDB LDAP clients on the is that the object class must be auxiliary.

The entry must contain the alias object class containing the attribute aliasedObjectName. For example, alias.

Note: As the alias object class is a structural class, the <identity type attribute> can be added using the extensibleObject object class. If an auxiliary object class is present in an entry the entry can optionally hold any attribute. The list of allowed attributes for this class is implicitly the set of all known server attributes.

Following is an example of MSISDN alias:

```
"MSISDN =<MSISDNnumber>, dc = msisdn, ou = identities, <CUDB root entry>"
```

The identity type attribute name must be the same as the identity type. In the above example, the identity type attribute name MSISDN matches the identity type msisdn.

3.1.4 Multi Service Consumer and Association Data

Specific data for each Multi Service Consumer (MSC) can be placed under separate entries identified by rdn: "mscId = <msc identifier>". Entries under this level are created for the services the MSC uses.



Association data specific entries are identified by `rdn: "assocId=<associ identifier>"`. Similarly to the MSCs, entries under this level are created for the services the MSC uses.

Figure 3 provides an example of the MSC and association data.

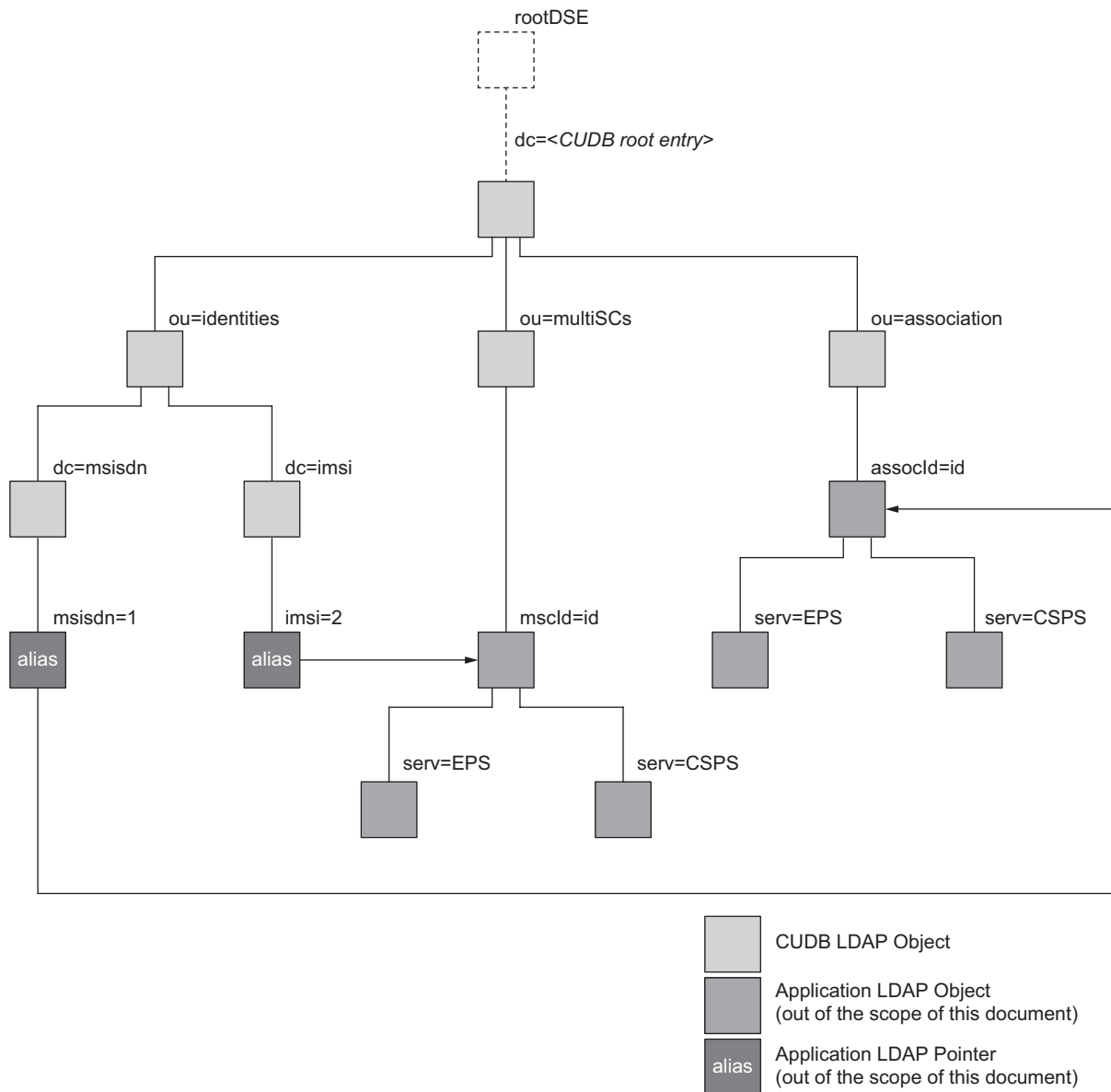


Figure 3 Multi Service Consumer and Association

The following MSC data entries are defined in CUIDB:

- MSC container: "ou = multiSCs,<CUIDB root entry>"
- MSCs: "mscId = <mscId>, ou=multiSCs, <CUIDB root entry>"



- MSC data for services can be placed under `mscId` entry in `"serv=<service>, mscId = <mscId>, ou=multiSCs, <CUDB root entry>"`, where `<service>` can take any value. For example, CSPPS, Auth and so on.

The following association data entries are defined in CUDB:

- Association container: `"ou = associations, <CUDBroot entry>"`
- Associations: `"assocId = <assocId>, ou=associations, <CUDB root entry>"`
- Association data for services can be placed under `assocId` entry in `"serv=<service>, assocId = <assocId>, ou=associations, <CUDB root entry>"`, where `<service>` can take any value. For example, CSPPS, EPS and so on.

Container entries `"ou = multiSCs, <CUDB root entry>"` and `"ou = associations, <CUDBroot entry>"` cannot be deleted, as they are CUDB DE parent entries. For more information, refer to the *DEs* section of *CUDB LDAP Data Access, Reference* [1].

3.1.5 MSC Common Data

MSC common data contains data shared by a group of MSCs, placed under the container level entry `"ou = mscCommonData"`. Entries under this level are created for the services the MSC uses. Each entry is referenced by all the MSCs that share this common data using LDAP alias.

Figure 4 shows an example of MSC common data.

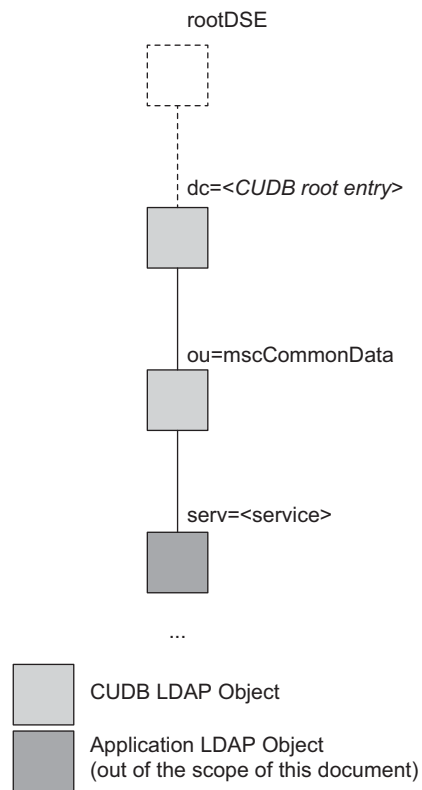


Figure 4 *MSC Common Data*

The following MSC common data entries are defined in CUDB:

- MSC common data: "ou=mscCommonData, <CUDB root entry>"

MSC common data for different services is placed under the mscCommonData entry in specific MSC common entry data placed under "serv=<service>,ou=mscCommonData, <CUDB root entry>".

3.1.6

Service Common Data

Service common data contains data shared on application FE level. Entries under this level are created for different services.

Figure 5 shows an example of service common data.

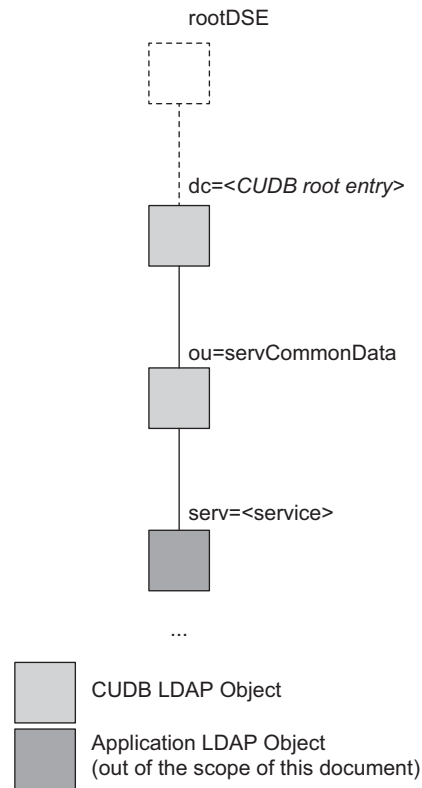


Figure 5 Service Common Data

The following entries are defined in CUDB:

- Service common data: "ou=servCommonData, <CUDB root entry>"

MSC common data for services can be placed under mscCommonData entry.

- Service specific common data is placed under "serv=<service>,ou=servCommonData, <CUDB root entry>"

3.1.7

CUDB Application Counters Directory Information Tree

CUDB provides read-only access through the LDAP interface to the set of Application Counters by exposing those counters in a separate branch in the LDAP DIT:

- Root Entry: "ou=ApplicationCounter"
- Application Counters Group entries that host no data: "dn: cn=<Counters Group Name>, ou=ApplicationCounter"
- Application Counters entries are organized below their respective Application Counters Group parent entry, and host the counter name and



value: "dn: cn=<Application Counter Name>,cn=<Counters Group Name>, ou=ApplicationCounter"

For more information, refer to *CUDB Application Counters*, Reference [4].

Figure 6 shows the Application Counters DIT.

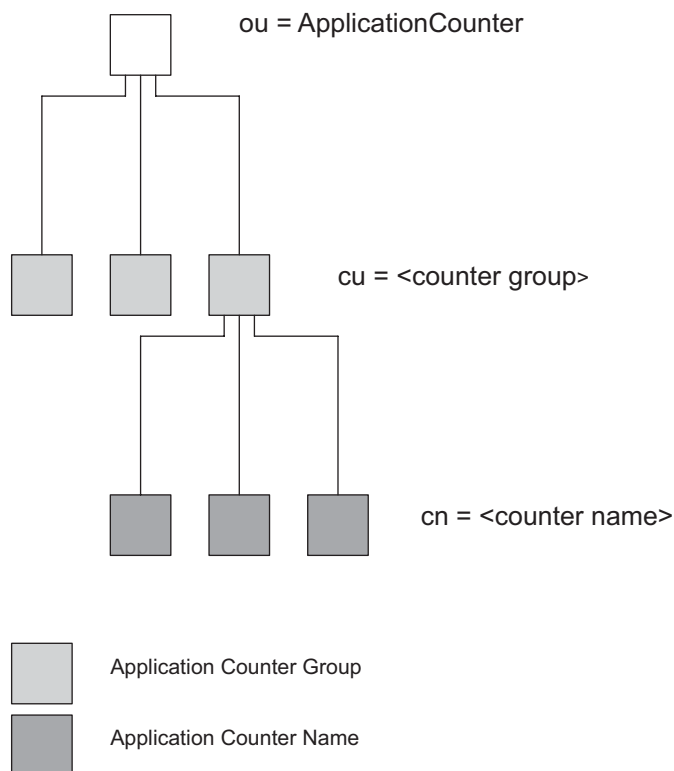


Figure 6 Application Counters DIT

The new LDAP FE Back End (BE) for counters fetches the value of the column *<Application Counter Name>* from the table *<Counters Group Name>*, and puts it in the generic value LDAP attribute. To fetch a counter value, the following LDAP search operation is used:

- Basedn: "ou=ApplicationCounter, cn=<CountersGroupId>, cn=<CounterId>"
- Scope: any, base/one/subtree will return the same.
- Filter: "objectclass=*" or "cn=<CounterId>" or "CounterValue=*" can be used.

An example of an LDAP search operation is shown below:



```
# ldapsearch -Y CUDB-CRYPTO -h pl_2_3 -U manager -W -b "cn=NSUBSCNT,cn=GRP_HLRSUBS,ou=ApplicationCounter" //
-s base # extended LDIF
#
# LDAPv3
# base <cn=NSUBSCNT,cn=GRP_HLRSUBS,ou=ApplicationCounter> with scope baseObject
# filter: (objectclass=*)
# requesting: ALL
#
# NSUBSCNT, GRP_HLRSUBS, ApplicationCounter
dn: cn=NSUBSCNT,cn=GRP_HLRSUBS,ou=ApplicationCounter
objectClass: ApplicationCounter
cn: NSUBSCNT
CounterValue: 3924600
# search result
search: 2
result: 0 Success
# numResponses: 2
# numEntries: 1
```

Note: A prompt to enter the LDAP root user password is shown.

3.1.8 dsgLocks

This LDAP DIT branch is used primarily by the CUDB system to allocate temporarily information during internal CUDB subscribers reallocation procedures. For more information, refer to *CUDB Subscription Reallocation*, Reference [6]. Typically, external client applications do not need to read or modify data on this subtree. The entry has the following format: "ou=dsgLocks,<CUDBroot entry>".

Figure 7 shows an example of the dsgLocks branch.

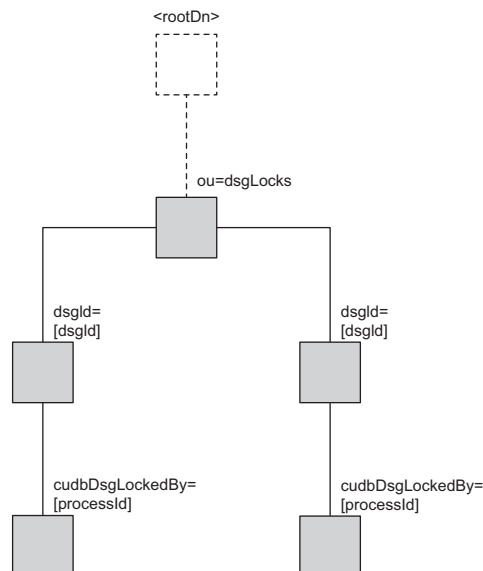


Figure 7 dsgLocks

The following dsgLocks entries are defined in CUDB:

- dsgLocks container: "ou=dsgLocks,<CUDBroot entry>"



- `dsgIds: "dsgId=<dsgId>,ou=dsgLocks,<CUDBroot entry>"`
- `cudbDsgLockedBy entries: "cudbDsgLockedBy=<processId>,dsgId=<dsgId>,ou=dsgLocks,<CUDBroot entry>"`

3.2 LDAP Views

Note: The LDAP Data Views requires function the Application Facilitator Value Package.

CUDB supports LDAP data views to support flexible data organization. If an LDAP user has an LDAP view assigned to it, then the user will access the data through that view.

Each LDAP data view has its own LDAP schema in the CUDB, building a custom DIT and a mapping file, describing how the virtual entries of the specific view are constructed from the core DIT attributes. The CUDB LDAP Data Views allows the data to be shown in different DITs accessible through the LDAP interface.

For more information, refer to *CUDB LDAP Data Views*, Reference [2].

3.2.1 Concepts

The following list contains the LDAP Data Views concepts:

- **Core DIT:** The current way the data stored in the database cluster is accessible through the LDAP interface in CUDB.
- **Real entry:** An entry in the CUDB core DIT.
- **LDAP Data View (View):** A configurable, tree-like alternative representation of the stored database cluster data. Entries are defined from attributes mapped from entries the core DIT.
- **Virtual entry:** An entry in a view, the attributes of which are mapped from one or several real entries.
- **One-to-one mapped entries:** Fully writable virtual entries that have a direct matching entry in the core DIT, with the same content as their core DIT counterparts.





4 Attributes and Object Classes

The following sections describe CUDB attributes and object classes defined in the default LDAP schema.

4.1 Attributes

Table 1 lists the attributes defined by CUDB, both for internal use and for the applications to store data inside CUDB.

Table 1 Attributes

Attribute Name	Description	Attribute Object Identity	Value Range	Example
assocId	A directory string type, single-value attribute that identifies an association.	1.3.6.1.4.1.193.169.2.51	1 - 32 characters	123456@msim.cudb
CDC	LDAP attribute defined in the base CUDB schema (cudb.schema file), which has a customized semantic and especial handling in CUDB for supporting the UDC application FE CDC mechanism. ⁽¹⁾	1.3.6.1.4.1.193.169.2.102	0-65535	23
CounterValue	The value retrieved from the Application Counters database.	1.3.6.1.4.1.193.169.2.401	1 - 32 characters	4201337
CUDBNode	The CUDB node (needed for LDAP proxy support).	1.3.6.1.4.1.193.169.2.101	Positive integer identifying a CUDB node. ⁽²⁾	1
DSUnitGroup ⁽³⁾	IA5 string type, single-value attribute identifying the DS UnitGroup where the data related to the DE is to be allocated. If this attribute is not included in an add operation, the system will automatically supply it with the correct value and it should not be modified afterwards. Data Store (DS) UnitGroup has precedence over the ZoneId attribute if both are included and has also precedence over a possible distribution algorithm included in a dynamic library that overrides the default distribution algorithm. This attribute can be modified when a reallocation procedure is performed. ⁽⁴⁾	1.3.6.1.4.1.193.169.2.100	Positive Integer identifying a DSG in CUDB. If the DSG is specified when provisioning, it must be a positive integer without any leading zeros. ⁽²⁾	1
ei	Directory string type, single-value attribute identifying an extensible entry.	1.3.6.1.4.1.193.169.2.300	1 - 32 characters	GPRS
mscId	Directory string type, single-value attribute identifying the MSC.	1.3.6.1.4.1.193.169.2.52	1 - 32 characters.	123456
serv	IA5 string type, single-value attribute identifying the service.	1.3.6.1.4.1.193.169.2.2	1 - 32 characters	CSPS



Table 1 Attributes

Attribute Name	Description	Attribute Object Identity	Value Range	Example
ZoneId ⁽³⁾	Integer type, single-value attribute identifying the zone the MSC belongs to. In MSC-related search filters, this attribute must be used with caution. If the attribute is not included in the add operations of the MSCs or if it was modified after the add operation, search results can be misleading. This attribute can be modified when a reallocation procedure is performed. ⁽⁴⁾	1.3.6.1.4.1.193.169.2.105	0-65535	1
dsgId	IA5 string type, single-value attribute identifying the DSG the entries of which have been locked during the <code>cudbReallocate</code> procedure.	1.3.6.1.4.1.193.169.2.304	3 characters.	1
cudbDsgLockedBy	IA5 string type, single-value attribute identifying the blade that started the <code>cudbReallocate</code> procedure and locked the necessary entries.	1.3.6.1.4.1.193.169.2.305	255 characters	SC_2_1
cudbDsgLockedDn	Attribute indicating distribution entries that have been locked during the <code>cudbReallocate</code> procedure.	1.3.6.1.4.1.193.169.2.306	Maximum allowed size for a DN in CUDB	mscId=1000,ou=multiSCs,<CUDB root entry>
expires	Indicates the date and time when the lock of the CUDBDsgLock entry expires.	1.3.6.1.4.1.193.169.2.113	1-255	201707271032Z
cudbBulkTarget	Special type of add operation that passes multiple LDAP operations at the same time.	1.3.6.1.4.1.193.169.2.450	2 characters	PL
cudbLdif	ldif data to be processed for deleting entries in LDAP FE.	1.3.6.1.4.1.193.169.2.451	1-512kB	Binary ldif stream
cudbLdifprov	ldif data to be processed for adding entries in LDAP FE.	1.3.6.1.4.1.193.169.2.452	1-512kB	Binary ldif stream

(1) For the CDC mechanism to work all LDAP clients accessing the entry must use the CDC attribute using the same option from those described in Section 6.1 on page 35. The CDC attribute is defined in the schema as a multi-value integer type to support providing several values in the LDAP write requests, although one only value is written and persistent in the database. In practice, the CDC attribute is stored as single-value, but handled as multi-valued while processing LDAP requests. Therefore, it should be defined as multi-valued.

(2) For more information about CUDB node configuration, refer to *CUDB Node Configuration Data Model Description*, Reference [3]

(3) This attribute must be defined in any object class included in the DE.

(4) For more information, refer to *CUDB Multiple Geographical Areas*, Reference [7].

4.2 Object Classes

An object class identifies a set of attributes in an entry in the DIT. Table 2 shows the LDAP object classes used in CUDB.

For more information on syntaxes and matching rules, refer to [RFC 4517 LDAP: Syntaxes and Matching Rules](#), Reference [11].



Table 2 Object Classes

Object Class Name	Description	Object Class Identity	Required Attributes	
ApplicationCounter	Structural object class storing the application counter value and name retrieved from the database.	1.3.6.1.4.1.193.169.1.202	cn CounterValue	Mandatory Mandatory
CounterGroup	Structural object class storing the name of an application counter group configured in CUDB.	1.3.6.1.4.1.193.169.1.201	cn	Mandatory
CUDBAssociation	Structural object class containing the attributes that handle an association and is included in the "ou=associations" container entry.	1.3.6.1.4.1.193.169.1.12	assocId ZoneId DSUnitGroup	Mandatory Optional Optional
CUDBCollisionDetection	Auxiliary object class containing the CDC attribute. Can be included in any container entry requiring collision control.	1.3.6.1.4.1.193.169.1.7	CDC	Optional
CUDBDcObject	Structural object class containing the dc attribute used by identity parent entries.	1.3.6.1.4.1.193.169.1.4	dc	Mandatory
CUDBExtensibleObject	Auxiliary object class handling addition of new entries to the current one.	1.3.6.1.4.1.193.169.1.3	ei	Optional
CUDBMultiserviceConsumer	It is a structural object class containing the attributes handling an MSC and is included in the ou="multiSCs" container entry.	1.3.6.1.4.1.193.169.1.11	mscId ZoneId DSUnitGroup	Mandatory Optional Optional
CUDBService	Structural object class containing an attribute that identifies a service and may be used in entries containing only other auxiliary object classes.	1.3.6.1.4.1.193.169.1.5	serv	Mandatory
CUDBServiceAuxiliary	Auxiliary object class containing an attribute that identifies a service and may be used in entries containing another structural object class.	1.3.6.1.4.1.193.169.1.6	serv	Mandatory
CUDBLockDistribution	Structural object class describing the distribution entry responsible for handling the locked entries of the cudbReallocate procedure.	1.3.6.1.4.1.193.169.1.301	dsgId DSUnitGroup	Mandatory Mandatory
CUDBDsgLock	Structural object class describing the entries in the DSG that hold the list of the entries locked for reallocation.	1.3.6.1.4.1.193.169.1.303	cudbDsgLockBy cudbDsgLockEdDn expires	Mandatory Optional Optional
CUDBBulk	Structural object class used when reallocating entries in bulk operations.	1.3.6.1.4.1.193.169.1.450	cudbBulkTarget cudbLdif cudbLdifprov	Mandatory Optional Optional



4.3 Operational Attributes

The CUDB LDAP server supports the following operational attributes, detailed in Table 3:

- `structuralObjectClass`.
- `createTimestamp`.
- `modifyTimestamp`.

For more information about these attributes, refer to [RFC 4512 LDAP: Directory Information Models](#), Reference [12].

Table 3 Operational Attributes

Attribute Name	Description	Attribute Object Identity	Value Range	Value Example
<code>structuralObjectClass</code>	Attribute indicating the structural object class of the entry.	1.3.6.1.4.1.1466.115.121.1.38	Any string identifying an object class	<code>structuralObjectClass: ImsImpu</code>
<code>createTimestamp</code>	Attribute storing the creation time of the entry in UTC format. ⁽¹⁾	1.3.6.1.4.1.1466.115.121.1.24	Any timestamp in GeneralizedTime format	<code>createTimestamp: 20140412050335Z</code>
<code>modifyTimestamp</code>	This attribute storing the last modification time of the entry in UTC format. ⁽¹⁾ If the entry has not been modified, the timestamp will be equal to the value of the <code>createTimestamp</code> attribute.	1.3.6.1.4.1.1466.115.121.1.12	Any timestamp in GeneralizedTime format	<code>modifyTimestamp: 20140329090437Z</code>

(1) These attributes may not be returned for all entries, check *CUDB LDAP Data Access*, Reference [1] for more information.

The following CUDB-specific operational attributes also exist:

- `nodeID`: the ID of the node from where the entry was returned, relevant in case of proxied requests.
- `entryDS`: the DS from which the entry was read.

4.4 Limitations

The limitations on attributes and object classes used in CUDB are the following:

- The maximum number of object classes defined per entry is 100.
- Inheritance between object classes is not supported.
- An object class cannot have more than 121 attributes.
- Object class capacity is limited to 8 KB. This limit does not include possible binary large objects (BLOBs) in the object class.



- Attributes can store up to 4 KB, except for octet string attributes that can hold up to 4 GB.

Note: The default maximum size for `octetString` and `string` attributes in CUDB is 255 octets. All attributes larger or significantly smaller may indicate the minimum upper bound for the size between curly brackets “{ }” after the syntax Object Identifier (OID). For more information, refer to [RFC 4512 LDAP: Directory Information Models](#), Reference [12].

DirectoryString and DN attributes use 3 bytes per character.

- DEs cannot contain multi-valued attributes.
- The depth of the DIT, that is, the number of sublevels including the root is 12.
- The RDNs of any DN are limited to 80 unicode characters.
- The maximum length of any DN is 512 unicode characters.
- Alias de-referencing in `subtree` or `onelevel` searches is not supported if the aliases are in subentries under the base DN.
- Aliases in entries under DEs that point to other entries under DEs are supported only if both are allocated in the same DS Unit Group (DSG). For more information, refer to *CUDB LDAP Data Access*, Reference [1].
- The maximum length of BASE64 encoded LDAP Data Interchangeable Format (LDIF) for LDAP extended operation is limited to 512 KB. This equals to 384 KB limit for decoded content.





5 CUDB DIT Entries

The following sections describe the predefined entries of the CUDB DIT created at installation time. It also contains entries that are expected to be created under the predefined entries using LDAP schema.

5.1 CUDB Admin Entry

The CUDB admin container entry holds the data for CUDB LDAP users. Table 4 shows the CUDB admin entry.

Table 4 CUDB Admin Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
ou=admin, <CUDBroot entry>	<ul style="list-style-type: none"> top organizationalUnit 	ou	Fixed	"admin"

5.2 Identities Container Entry

The identities container entry holds the identities for the MSCs grouped by identity domains. Table 5 shows the identities container entry.

Table 5 Identities Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
ou= identities, <CUDB root entry>	<ul style="list-style-type: none"> top organizationalUnit 	ou	Fixed	"identities"

5.3 Multi Service Consumer Container Entry

The MSC container entry holds the entries containing the data for all the MSCs in CUDB. Table 6 shows the MSC container entry.

Table 6 Multi Service Consumer Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
ou= multiSCs, <CUDB root entry>	<ul style="list-style-type: none"> top organizationalUnit 	ou	Fixed	"multiSCs"



5.3.1 Multi Service Consumer Data Entry

The MSC data container entry contains an MSC. Sub-entries under this level are created for the services the MSC uses. Table 7 shows an MSC consumer data entry.

Table 7 Multi Service Consumer Data Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
mscId= <mscId>, ou=multiSCs, <CUDB root entry>	<ul style="list-style-type: none">topCUDBMultiServiceC onsumer	mscId	Variable	"123456"
		zoneId	Variable	1
		DSUnitGroup	Variable	1

5.3.2 Multi Service Consumer Data Service Entry

The MSC data service entry is the container for the service-specific multi service consumer data in CUDB. Table 8 shows an MSC data service entry.

Table 8 Multi Service Consumer Data Service Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
serv=<service>,mscId = <mscId>, ou=mul tiSCs, <CUDB root entry>	<ul style="list-style-type: none">topCUDBService	serv	Variable	"CSPS"

5.4 Associations Container Entry

The associations container entry is the container for the entries containing data for all the associations in CUDB. Table 9 shows the associations container entry.

Table 9 Associations Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
ou= associations,<CU DB root entry>	<ul style="list-style-type: none">toporganizationalUnit	ou	Fixed	"associations"

5.4.1 Association Data Entry

An association data entry contains an association. Sub-entries under this level are created for the services the association uses. Table 10 shows an association data entry.



Table 10 Association Data Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
assocId= <assocId>, ou=associations, <CUDB root entry>	<ul style="list-style-type: none"> top CUDBAssociation 	assocId	Variable	"123456@msim.cudb"
		ZoneId	Variable	"1"
		DsUnitGroup	Variable	"1"

5.4.2 Association Data Service Entry

An association data service entry is the container for the service-specific association data in CUDB. Table 11 shows an association data service entry.

Table 11 Association Data Service Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
serv=CSPS, assoc Id=<assocId>, ou= associations, <CUDB root entry>	<ul style="list-style-type: none"> top CUDBService 	serv	Variable	"CSPS"

5.5 Multi Service Consumer Common Data Container Entry

An MSC common data container entry is the container for MSC common data in CUDB. Table 12 shows an MSC common data container entry.

Table 12 Multi Service Consumer Common Data Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
ou=mscCommonData, <CU DB root entry>	<ul style="list-style-type: none"> top organizationalUnit 	ou	Fixed	"mscCommonData"

5.5.1 Multi Service Consumer Common Data Service Entry

An MSC common data service entry is the container for the service-specific service common data in CUDB. Table 13 shows an MSC common data service entry.

**Table 13** Multi Service Consumer Common Data Service Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>serv=<service>,ou=mscCommonData,<CUDB root entry></code>	<ul style="list-style-type: none">• top• CUDBService/CUDB ServiceAuxiliary	<code>serv</code>	Variable	"CSPS"

5.6 Service Common Data Container Entry

A service common data container entry is the container for the service common data in CUDB. Table 14 shows service a common data container entry.

Table 14 Service Common Data Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>serv=<service>,ou=servCommonData,<CUDB root entry></code>	<ul style="list-style-type: none">• top• organizationalUnit	<code>ou</code>	Fixed	"servCommonData"

5.6.1 Service Common Data Service Entry

A service common data service entry is the container for the service specific related service common data in CUDB. Table 15 shows a service common data service entry.

Table 15 Service Common Data Service Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>ou=servCommonData,<CUDB root entry></code>	<ul style="list-style-type: none">• top• CUDBService/CUDB ServiceAuxiliary	<code>serv</code>	Variable	"CSPS"

5.7 dsgLocks Container Entry

The `dsgLocks` container entry is the container for the entries containing data for all of the `cudbReallocate` locking entries. Table 16 shows the `dsgLocks` container entry

Table 16 dsgLocks Container Entry

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>ou=dsgLocks,<CUDB root entry></code>	<ul style="list-style-type: none">• top• organizationalUnit	<code>ou</code>	Fixed	"dsgLocks"



5.7.1 dsgLocks Data Entry

A `dsgLocks` data entry contains a `dsgId` where the entries holding the distribution entries that are locked during the `cudbReallocate` procedure will be placed. Table 17 shows a `dsgLocks` data entry.

Table 17 *dsgLocks Data Entry*

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>dsgId=<dsgId>,ou=dsgLocks,<CUDB root entry></code>	<ul style="list-style-type: none"> top CUDBLockDistribution 	<code>dsgId</code>	Variable	1
		<code>DSUnitGroup</code>	Variable	1

5.7.2 dsgLocks Data Distribution Entry

A `dsgLocks` data distribution entry contains information about the distribution entries locked by the specified blade. Table 18 shows a `dsgLocks` data distribution entry.

Table 18 *dsgLocks Data Distribution Entry*

Entry Name	Object Class Name	Attributes		
		Name	Value Type	Value Example
<code>cudbDsgLockedBy=<processId>,dsgId=1,ou=dsgLocks,<CUDB root entry></code>	<ul style="list-style-type: none"> top CUDBDsgLock 	<code>cudbDsgLockedBy</code>	Variable	SC_2_1
		<code>cudbDsgLockedDn</code>	Variable	<code>mScId=1000,ou=multiSCs,<CUDB root entry></code>
		<code>expires</code>	Variable	201707271032Z



6 Operations

The following LDAP operations are used to handle CUDB data:

- Bind
- Add
- Delete
- Modify
- Search
- Unbind

The following LDAP extended operations, detailed in Table 19, are used to handle CUDB data:

- Ldapbundled
- Ldapbundledadd

Table 19 LDAP Extended Operations

Operations	OID	Description
Ldapbundled	1.3.6.1.4.1.193.169.4.1	Generic bundled write request
Ldapbundledadd	1.3.6.1.4.1.193.169.4.2	Provisioning bundled write request

The LDAP Data Views function uses the following LDAP operations when accessing data through view:

- Modify-add: add an attribute to a virtual entry, resulting in a new attribute in the existing real entry.
- Modify-delete: delete an optional attribute from a virtual entry, resulting in a deleted existing attribute in the real entry.
- Modify-replace: replace an attribute in a virtual entry, resulting in a replaced attribute in the existing real entry. If the attribute does not exist, the request is handled as modify-add. If no value is provided, the request is handled as modify-delete.
- Search: searches involving data from both Processing Layer Database (PLDB) and DSG are not supported.

When accessing data through LDAP data views, special entries can be used, called one-to-one mapped entries.

One-to-one mapped entries are real entries with the exact the same contents as the virtual entries. They can be created, deleted, and their contents modified through views without any restriction other than the ones that apply to a real entry.

Note: The LDAP Data Views function requires the Application Facilitator Value Package.

There are certain special provisions in CUDB regarding standard LDAP operations.

- CUDB does not support including the abstract `objectClass top` in LDAP modify operations. The abstract `objectClass top` in modify operations is optional, according to LDAP RFCs. In such case, CUDB returns the following error:

```
err=80, msgText=Object class not configured in  
OBJECT_CLASSES table
```

- Alias de-referencing in LDAP modify operations for primary identities is a non-standard behavior offered by CUDB. This behavior implies that the aliases found when resolving the target DN specified in an LDAP modify operation are always de-referenced if the original DN of the object to be modified contains a primary identity.

Two examples for alias de-referencing with a DN that contains a primary identity are as follows:

- `serv=csp,IMSI=123456789,dc=imsi,ou=identities,<suffix>`
- `serv=csp,MSISDN=346162681,dc=msisdn,ou=identities,<suffix>`

A CUDB LDAP modify operation on these two entries will result in the modification of the two entries being referenced by the **IMSI=123456789** and **MSISDN=346162681** aliases. It means that both, **mscld=id** and **assocld=id** are going to be modified through their corresponding aliases, as depicted in Figure 8:

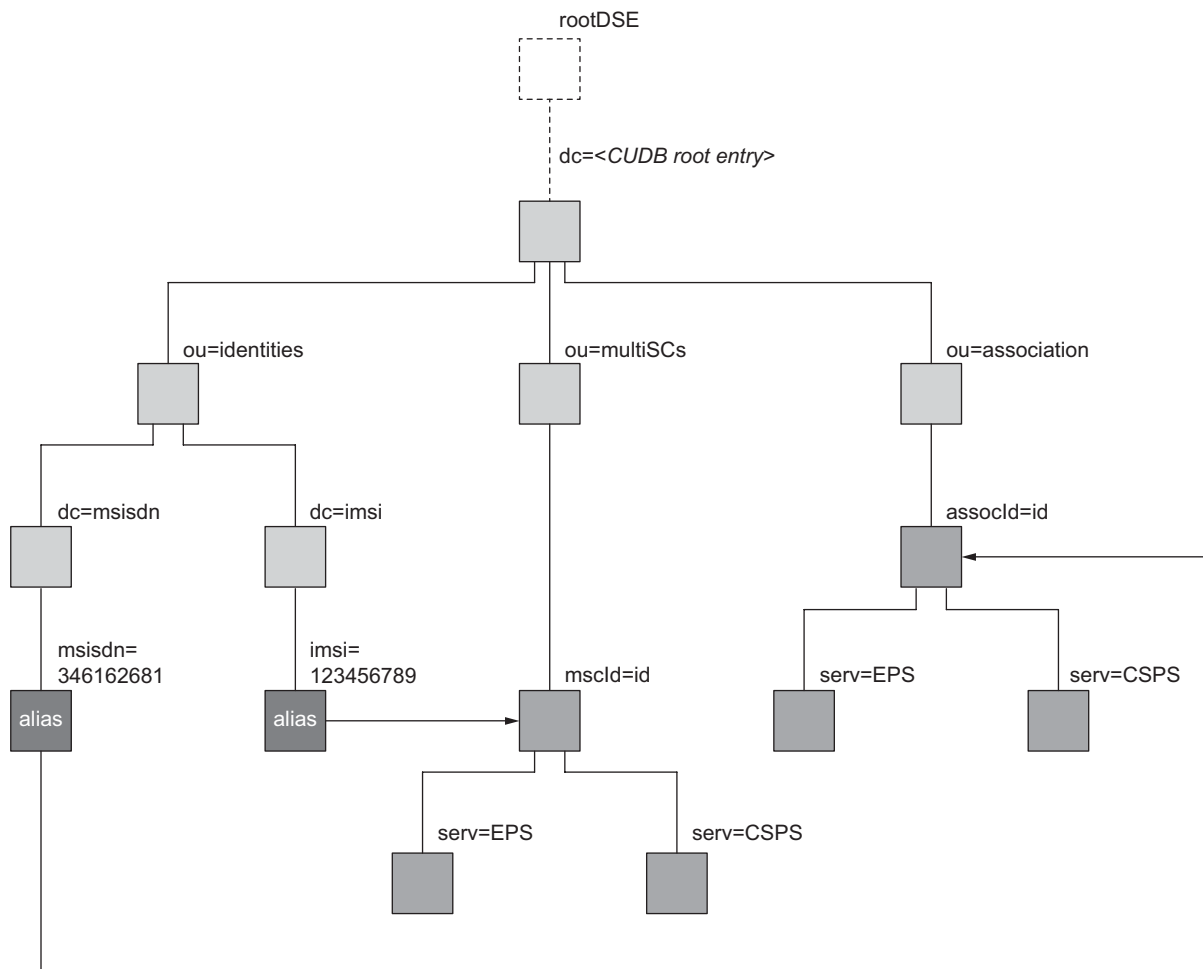


Figure 8 LDAP Modify Operation for Primary Identities

A standard behavior of the LDAP modify would not de-reference the query and in the example above, no modification would take place in **mscld=id** and **assocld=id**. In such case, the LDAP modify query returns an error saying that the required entry does not exist.

6.1 CDC

Client Applications can have the Collection Detection Counter (CDC) attribute in LDAP entries to avoid race conditions when several clients access LDAP entries at the same time. An example for such race condition is LDAP client B performing a write operation on an entry between a previous read operation and a subsequent write operation by LDAP client A on the same entry. If both clients use the same CDC method from the options described below, the first client can assert that no other client accessed the entry after it read that entry.

There are two alternative ways to implement collisions detection using the CDC attribute within an entry:



6.1.1 Option 1

1. Client A reads the LDAP entry, along with its CDC value.
2. Client A modifies the LDAP entry and the CDC attribute the following way:
 - a. Add multiple new values with the add attribute operation to the multi-valued CDC attribute, the values ranging from `<CDC_read_value>+1` to `<CDC_read_value>+N`, N being the tolerance factor detecting up to N potential changes made by other clients in the time since the read was done.

The LDAP modify operation:

```
[add/modify/replace/delete operations on attributes belonging to the entry]
```

```
add: CDC: <CDC_read_value>+1
```

```
add: CDC: <CDC_read_value>+2
```

```
...
```

```
add: CDC: <CDC_read_value>+N
```

- b. In the same LDAP request, use a modify-replace operation on the CDC attribute CDC: `<CDC_read_value>+1`:

```
replace: <CDC_read_value>+1
```

- c. Perform any other add/modify/replace/delete operations in the same LDAP request on attributes in that LDAP entry.

The LDAP operation fails if one or up to N other LDAP clients changed the LDAP entry and its CDC value between the read and the write operations by client A. The way the LDAP modify operation is evaluated makes the CDC add operation fail if CDC already includes any of the values to add to and returns LDAP `errorCode=20 attributeOrValueExists`. If 1 to up to N other LDAP clients changed the LDAP entry between the read and the write operations by client A, CDC would also change to a value equal to one of the values in the add operations, `<CDC_read_value>+1` to `<CDC_read_value>+N`, specified by client A, and the write operation will not succeed.

If the add check passes, the CDC replace operation in the same request will successfully write one single value of the `<CDC_read_value>+1` into the CDC, and the rest of the add/modify/replace operations on the entry will go through.

Note: The `replace` operation within the LDAP `modify/write` request sets only one value of the multi-valued CDC attribute.

6.1.2 Option 2

1. Client A reads the entry, along with its CDC value.
2. Client A updates LDAP entry as needed, and at the end of the modify operation, includes a delete attribute operation on the read CDC value.



The LDAP modify operation:

```
[add/modify/replace/delete operations on attributes belonging to the entry]
delete: CDC: <CDC_read_value>
```

The LDAP operation fails if any other LDAP client changed the entry and the CDC value between the read and write operations by client A. The LDAP modify operation is evaluated so that the check if the CDC includes the value to be deleted is the first step. If any client changed the CDC to a newer value, then the delete attribute operation on the former value will fail.

In the same write request, use a modify-replace operation on the CDC attribute
 CDC: <CDC_read_value>+1:

```
replace: CDC: <CDC_read_value>+1
```

If the `delete` check passes, the `replace` operation in the same request writes the next value of the CDC.

Option 2 is recommended, as it works irrespective of the number of other LDAP clients that attempted to modify the LDAP entry between the read and the write operations by client A. It does not limit the number to N potential CDC changes.

Note: The `replace` operation within the LDAP `modify/write` request sets only one value of the multi-valued CDC attribute.

6.2 Base DN Transformation in LDAP operations over Authentication Data Operations (serv=Auth)

CUDB handles certain situations for application FEs uniquely, such as in requests from UDM applications, `baseDn` transformations are applied transparently to cover these use cases.

Therefore, when a search operation is received from any CUDB LDAP client with the following

```
"baseObject: dn: serv=Auth, IMSI =<IMSInumber>, dc =
imsi, ou = identities, <CUDB root entry>"
```

and alias de-referencing is equal to finding or always, the operation is interpreted as if the following

```
"baseObject: dn: IMSI =<IMSInumber>, serv=Auth, IMSI
=<IMSInumber>, dc = imsi, ou = identities, <CUDB root entry>"
was received instead.
```

The same applies for any LDAP modify operations for the above `baseObject`.



6.3 Performing LDAP Search Below the DE

The CUDB subscriber-centric data model, depicted in Figure 1, enables data consolidation through storing application service profiles right below a DE, in an `mscId` or an `asscoId`.

Client applications retrieve only their own service profiles data, that is, their own application data, below their `serv=<service>` entries) and are unaware of other populated profiles created at the same level below the DE.

The CUDB search engine is optimized for LDAP search operations that target individual entries that use the entry DN as search base object. Use this type of search for the best performance.

To enhance the performance of LDAP subtree searches, the CUDB search engine implements an index optimization for search operations that look through the LDAP DIT starting downwards from a base entry and fetch different LDAP entries. For more information, refer to the *LDAP Massive Search Indexes* section in *CUDB Application Integration Guide*, Reference [9].

To take full advantage of search index optimization, choose the best possible search base for the LDAP operation, such as the location in the directory tree from which the LDAP subtree search begins.

As a general recommendation, always start application subtree searches from a `serv=<service>` entry or any other entry below it as the search base object. Do not use start subtree searches starting from the DE, otherwise the scanning of every other service profiles and their child entries can cause significant performance drop.

In case of using subtree searches starting from the DE, those can be optimized and executed faster by configuring the Optimized Subtree Searches function. With that feature it is possible to narrow the scope of the subtree search down to a list of specified targets. For more information refer to the *Optimized Subtree Searches* section in *CUDB System Administrator Guide*, Reference [5].

6.4 Error Codes

The CUDB LDAP interface supports the LDAP result codes defined in the LDAP protocol specifications. For more information, refer to [RFC 4511 LDAP: The Protocol](#), Reference [13].

Table 20 shows the LDAP result codes in CUDB, describing the possible causes for each error and possible actions about how to handle these errors from the client application perspective.



Note: The "Description" provides the most common reasons why the corresponding error is received. This description does not provide all possible causes. The message diagnostic text may provide more specific information about the error.

For additional information on overload-related causes, mastership change, unavailability, or unreachability, refer to the following Facility Descriptions:

- *CUDB LDAP Data Access*, Reference [1]
- *CUDB High Availability*, Reference [8]

Table 20 CUDB LDAP Error Codes

Error Code	Error Type	CUDB Error Description	Possible Client Actions
3	Time limit exceeded	<p>In addition to the time limit set by the client exceeded, CUDB adds the following cause:</p> <ul style="list-style-type: none"> • Proxy timeout in a search operation, caused by exceeding the time limit of an LDAP client, if it is lower than the LDAP FE time limit. 	If the problem is due to proxy timeout, the LDAP client can retry the request.
11	Administrative limit exceeded	<p>In addition to LDAP causes, CUDB adds the following possible causes due to administrative limits, such as size limit and time limit:</p> <ul style="list-style-type: none"> • Proxy timeout in add, delete, or modify operations. • Proxy timeout in a search operation caused by exceeding the LDAP FE time limit. 	If the problem is due to proxy timeout, the LDAP client can retry the request.



Table 20 CUDB LDAP Error Codes

Error Code	Error Type	CUDB Error Description	Possible Client Actions
51	<ul style="list-style-type: none">• Busy• Overload or congestion in the processing layer at CUDB node level.• Administrative limit exceeded.	<ul style="list-style-type: none">• The LDAP FE or the PLDB in a specific CUDB node are experiencing high traffic load, above their planned capacity. The node implements overload protection and rejects incoming operations with this error code if they are unable to process them.• The maximum number of threads that can access the local PLDB database in an LDAP FE has been reached.• The probable cause of reaching the maximum number of PLDB threads is an overload in the PLDB.• There is only one unique DSG deployed in a CUDB system and it is overloaded. When the master replica of the unique DSG in the CUDB system is overloaded, the LDAP FE on the same node returns error code 51 directly.• In specific condition, short bursts of EC51 (temporal and short error rates <0,01%) may be observed, although the CUDB is not overloaded. In case the network quality is bad between client application and CUDB (such as high packet loss rate and high network latency), and if client TCP connections are carrying intense throughput (high rate of TPS per connection), a small number of operations could be rejected with EC51 related to TCP congestion mechanisms. Regarding the overall incoming traffic, the error rate produced by those bursts of EC%1 rejection may be considered negligible.	<ul style="list-style-type: none">• The LDAP client can regulate the traffic sent towards the CUDB node until the congestion situation is resolved, for example, until no more errors than 51 are received.• The LDAP client can retry the request, for example in case of provisioning-related requests.
52	<ul style="list-style-type: none">• Unavailable• The CUDB node is unavailable to process requests.	<p>The CUDB node does not have enough available resources, or a failure occurred in a critical component. The error can be caused by one of the below scenarios:</p> <ul style="list-style-type: none">• Minority scenarios, where no master replica for any DSG is reachable.• Symmetrical Split scenarios, for search operations performed by provisioning users or search operations with <code>ReadMode control</code> value 0 on the side of the split not hosting the master PLDB replica. For more information about control, see Section 6.5.1 on page 41.	<p>Reconnect to another CUDB node, that is, execute a failover towards another CUDB node.</p>



Table 20 CUDB LDAP Error Codes

Error Code	Error Type	CUDB Error Description	Possible Client Actions
53	Unwilling to perform.	<ul style="list-style-type: none"> Queries intended for the Processing Layer (PL) or the DS partition where the master is unreachable, or no replica is available. Single Level and wholeSubtree scopes are not supported in this part of the LDAP tree. Operation is not supported within namingContext. 	<ul style="list-style-type: none"> Do not perform immediate and frequent retries when this error appears. Increase the retry delay for the application, network, or terminal retries.
80	<ul style="list-style-type: none"> Other error. A problem in an internal component or resource that prevents CUDB from successfully processing the request. Other parts of the system may be operating under normally and return successful responses. 	<ul style="list-style-type: none"> Data Store (DS) Unit overloaded. Remote CUDB node overloaded or not responding. Subscriber blocked due to reallocation (temporary error). Ongoing master change (temporary error). Application counters BE is busy. LDAP FE is restarting (temporary error). Maximum number of DS threads that can access to a local DSG replica. 	<p>A possible cause for this error is the temporal unavailability of an internal component or a temporary internal node or partition overload. Increase the delay between the application, network, or terminal retries for operations rejected by CUDB with error 80.</p> <ul style="list-style-type: none"> If the problem is due to the maximum number of threads being exceeded. To speed up system stabilization, increase the delay between application, network, or terminal retries for operations rejected by CUDB with error 11. If the problem is due to an internal CUDB component, the client cannot consistently handle the error. If possible, to speed up server stabilization, the client attempts to increase the delay between retries or lower the number of operations that are rejected from CUDB with error 80.

For more information on error codes, refer to *CUDB LDAP Data Access*, Reference [1].

6.5 LDAP Controls

LDAPv3 allows clients and servers to use controls as a mechanism for extending an LDAP operation. A control is a way to specify additional parameters as part of a request and a response. For more information, refer to [RFC 4511 LDAP: The Protocol](#), Reference [13].

6.5.1 ReadMode Control

The CUDB LDAP interface supports ReadMode control with Object ID=1.3.6.1.4.1.193.169.3.3, which is used to temporarily override the LDAP read preferences for PLDB and DSG.

The ReadMode LDAP control is not supported for LDAP write operations, such as modify, add, or delete. If the Extension Control criticality in the write request is set to `FALSE`, the ReadMode control is ignored and operation is processed normally. If the Extension Criticality is set to `TRUE`, the LDAP request will



receive an `err=12, msgText=critical extension is unavailable` message.

In case of distributed or massive searches, that is, search requests distributed to various DSGs, ReadMode control content is always ignored.

When ReadMode control is used, the recommended setting for criticality is `FALSE`.

For more information, refer to [RFC 4511 LDAP: The Protocol](#), Reference [13].

The ReadMode control has an ASCII string value, indicating a combination of `readModeInPL` and `readModeInDS` that is to be applied to particular LDAP search request.

Table 21 Supported ReadMode LDAP Control Values

Value	For PL	For DS
0	Master always	Master always
1	Local preferred	Master preferred

When a search request with ReadMode control is received, the read mode control values extracted take precedence over `readModeInDS` and `readModeInPL` values configured for the LDAP user.

6.6 CUDB LDAP Extended Operations

CUDB LDAP interface supports two additional LDAP extended operations that allows packing multiple write operations that are bundled into one single LDAP request.

Extended operations are defined by its OID identifier and they allow adding an LDIF payload with the multiple batched operations. There are two different operation types, which are used in different use cases as described in Table 19.

If any of the bundled operations fails, the entire LDAP operation will fail.

Generic write request operation allows executing batches of multiple add, modify, and delete operations.

Provisioning write request operation only supports batches of add operations.

For generic writes LDAP extended operation will return a “**matchedDN**” field (along with error code and description text) to point out on which entry in the batch the operation failed.

Data to be processed must comply with the following validation rules:

- All entries are targeting either one of PL or DS. No mix is allowed.
- When targeting a DS, all DS entries are under the same DE.



- If provisioning operation type, all entries must be siblings and/or in parent-child relationship.
- If provisioning operation type, the change type for all operations in LDIF file must be ADD.
- Alias dereferencing is not supported.
- Parent must be added before its children.
- Children in the LDIF must be deleted before the parent.
- Size of bulk data must not exceed 512 KB when BASE64 encoded (384KB when decoded).





Glossary

For the terms, definitions, acronyms and abbreviations used in this document, refer to *CUDB Glossary of Terms and Acronyms*, Reference [10].





Reference List

CUDB Documents

- [1] *CUDB LDAP Data Access*
- [2] *CUDB LDAP Data Views*
- [3] *CUDB Node Configuration Data Model Description*
- [4] *CUDB Application Counters*
- [5] *CUDB System Administrator Guide*
- [6] *CUDB Subscription Reallocation*
- [7] *CUDB Multiple Geographical Areas*
- [8] *CUDB High Availability*
- [9] *CUDB Application Integration Guide*
- [10] *CUDB Glossary of Terms and Acronyms*

Other Documents and Online References

- [11] *Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules, RFC 4517* <http://www.rfc-editor.org/rfc/rfc4517.txt>
- [12] *Lightweight Directory Access Protocol (LDAP): Directory Information Models, RFC 4512* <http://www.rfc-editor.org/rfc/rfc4512.txt>
- [13] *Lightweight Directory Access Protocol (LDAP): The Protocol, RFC 4511* <http://www.rfc-editor.org/rfc/rfc4511.txt>