

CUDB LDAP Data Views

FACILITY DESCRIPTION

Copyright

© Ericsson AB 2016. 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	Revision Information	1
1.3	Concepts	1
1.4	Target Groups	2
1.5	Prerequisites	2
1.6	Typographic Conventions	2
2	Overview	3
2.1	Prerequisites	3
2.2	Architecture	3
2.3	How to Use This Function	4
2.4	Example of a Virtual Tree	6
2.5	Principles of LDAP Data Views	9
2.6	Scaffolding Virtual Entries	10
2.7	One-to-one Mapped Entries	11
3	Description of Runtime Behavior	13
3.1	General Rules	13
3.2	Search Requests	14
3.3	Write Requests	17
4	Dependencies and Interactions	21
5	Operation and Maintenance	23
5.1	Configuration	23
5.2	Fault Management	23
5.3	Performance Management	23
5.4	Security	23
5.5	Logging	24
	Glossary	25
	Reference List	27





1 Introduction

This document describes the Lightweight Directory Access Protocol (LDAP) Data Views function in Ericsson Centralized User Database (CUDB). The purpose of LDAP Data Views is introduced, as well as the necessary concepts, configuration, and behavior with explanations to ensure full understanding of the function and how to use it.

The LDAP Data Views function supports views, which allow for customizable ways of viewing the data.

Note: The LDAP Data Views function can only be used if the Application Facilitator Value Package is available.

1.1 Document Purpose and Scope

The purpose of this document is to describe how the LDAP Data Views function works to allow their optimized use.

1.2 Revision Information

Rev. A This document is based on 15/155 34-HDA 104 03/9 and contains the following changes:

- Section 1 on page 1, Section 1.5 on page 2, and Section 5.1 on page 23: Updated restriction information.

1.3 Concepts

Core DIT (Directory Information Tree)

The base data representation of the information stored in CUDB. It consists of a hierarchical tree-like structure hosting LDAP entries identified by their Distinguished Names (DNs).

This base LDAP tree is used by default to access and provision the data through the LDAP interface, export stored data in LDIF format, configure Notification events on data objects, and is the LDAP data presentation used by CUDB internal clients or procedures using the LDAP protocol.

The LDAP Data Views function uses the core DIT as the main data reference when defining virtual entries.

Real entry An entry in the core DIT.



LDAP Data View (View)

A configurable alternative tree-like representation of the data stored in CUDB. Each LDAP view is a collection of virtual entries, composed of virtual object classes and attributes, arranged in a hierarchical structure that are associated with attributes and entries in the core DIT.

LDAP views are assigned to LDAP users by configuration.

Virtual entry

An entity in an LDAP Data View tree consisting of a collection of attributes and object classes that are associated with and mapped from one or multiple real entries in the core DIT. Virtual entries have no storage footprint, rather are building blocks to alternatively presenting the data already stored in the database in different tree structures.

Virtual attribute

An attribute of a virtual entry that is associated with an attribute of a real entry. A virtual attribute gets its value from the attribute in the associated real entry.

One-to-one mapped entries

Fully writable virtual entries that have a direct matching entry in the core DIT. They have the same content as their core DIT counterparts.

1.4 Target Groups

This document provides general information about the LDAP Data Views function and is intended for system administrators and operators.

1.5 Prerequisites

Users of this document must have an overall knowledge of the CUDB system and LDAP basics.

Note: The LDAP Data Views function can only be used if the Application Facilitator Value Package is available.

1.6 Typographic Conventions

Typographic conventions can be found in the following document:

- *Typographic Conventions*



2 Overview

CUDB is a member of the User Data Consolidation (UDC) product family. Consolidation stands for the following two concepts:

- 1 Storing data from multiple, diverse application Front Ends (FEs) in a single database.
- 2 Data shared by multiple application FEs need only be stored once in the database, saving storage space and guaranteeing consistency by eliminating the need to keep multiple copies of a single data point up-to-date.

LDAP repositories (databases) store data according to a specific hierarchical structure: a particular LDAP tree, built using particular data structures (object classes and attributes). This is definition of the core Distributed Information Tree (DIT). When integrating application FEs in CUDB, certain FEs can be configured to look for common data wherever it is stored, or to store their own data in a way that fits into the core DIT of CUDB.

Other application FEs cannot be configured in such a way and they require specific LDAP trees as well as expect their own and shared data to be stored in specific locations of that LDAP tree, according to specific schemas, that is, object classes and attribute names. These specific LDAP trees might differ from the core DIT in the locations and names of the relevant entries, resulting in data duplication and complicated data maintenance, thus weakening data consistency and economy of storage.

The LDAP Data Views function addresses this problem by displaying data already stored in the core DIT in a customized way. Using a custom LDAP tree in a data view that can use custom object classes and attributes promotes application integration and facilitates data consolidation, reducing storage requirements and eliminating data duplication.

2.1 Prerequisites

This document assumes the knowledge of the CUDB LDAP FE, the LDAP protocols, services, and information models.

2.2 Architecture

Not applicable.



2.3 How to Use This Function

The purpose of LDAP Data Views is to display and access existing data stored and presented in the core DIT of CUDB in different ways. It is possible to create a customized LDAP tree structure, composed of a custom definition of entries, object classes, and attributes. The flexibility, made possible by allowing the creation of new, completely different LDAP trees to display and access the stored data, makes the integration of new or legacy application data models in CUDB possible, without needing to duplicate data, migrate data from an existing data model to the core DIT, or both.

To integrate a new application that cannot use data in the structure it is stored in the core DIT, complete the following steps to configure a custom data view:

1. *This step is optional.* Extend the current core DIT with new, application-specific data not yet stored in the database. A schema update may be required to define new object classes and attributes. The new data is added, typically through provisioning, into the previously existing or the newly created entries in the core DIT. If a schema update is required, refer to *CUDB Node Schema Update*, Reference [1].
2. Define a new data view schema with its virtual object classes and virtual attributes wherein to arrange the data already stored in the core DIT, according to what the application FE requires. Object classes and attributes defined in a data view schema will be present inside the view.
3. Define a new, virtual LDAP tree, built from the root down, following the structure the application FE expects to see, defining the DN's for every virtual entry in the virtual tree. For an example, see Figure 1.
4. For every virtual entry in the virtual LDAP tree, define which object classes must (`STRUCTURAL`) or may (`AUXILIARY`) be included in them. These object classes may be one of the following types:
 - “real”: Present in the CUDB schema, irrespective of whether they are old or have just been added in Step 1.
 - “virtual”: Not present in the CUDB schema, but defined in a data view schema, that is, in the context of the LDAP view.
5. For every object class in every virtual entry in the virtual tree, define mapping rules that describe which attributes should be present, and from where in the core DIT (entry and attribute) to retrieve their value.

Steps 3, 4, and 5 must be performed in XML format, thus creating the mapping configuration file. The mapping configuration XML file defines the entries and the organization of the virtual tree. The hierarchy of the entries can be changed, with respect to the contents of core DIT.

Each LDAP Data View must have its own mapping configuration.



For details on how to create and where to store data view schemas and mapping configurations, refer to *CUDB LDAP Data Views Management*, Reference [2].

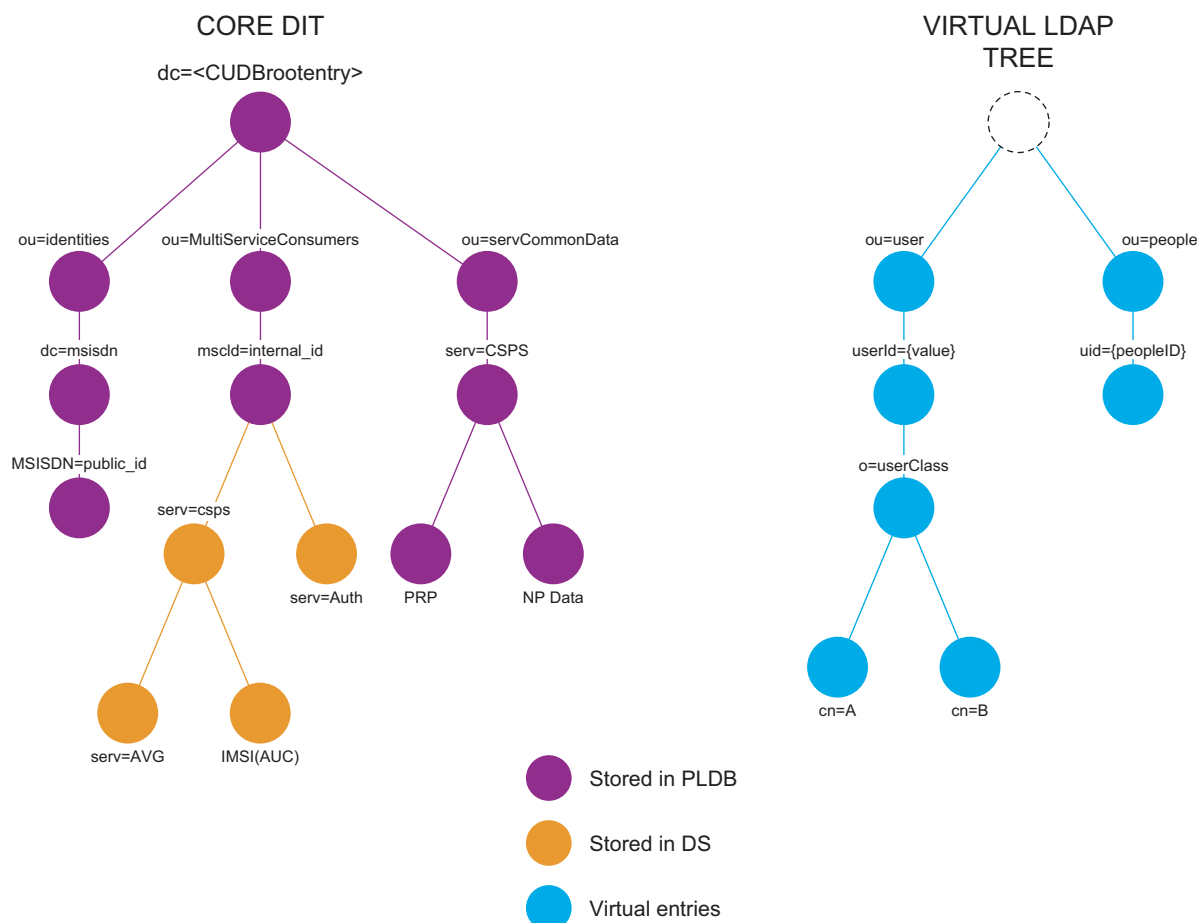


Figure 1 LDAP Core DIT Tree and LDAP View Tree Sharing the Same Stored Dataset

Once a custom data view is configured, it can be assigned to one or multiple LDAP users. If an LDAP user has been assigned a view, that LDAP user will stop seeing data in CUDB as it is stored in the core DIT and will see CUDB data according to the view it has been assigned.

2.4 Example of a Virtual Tree

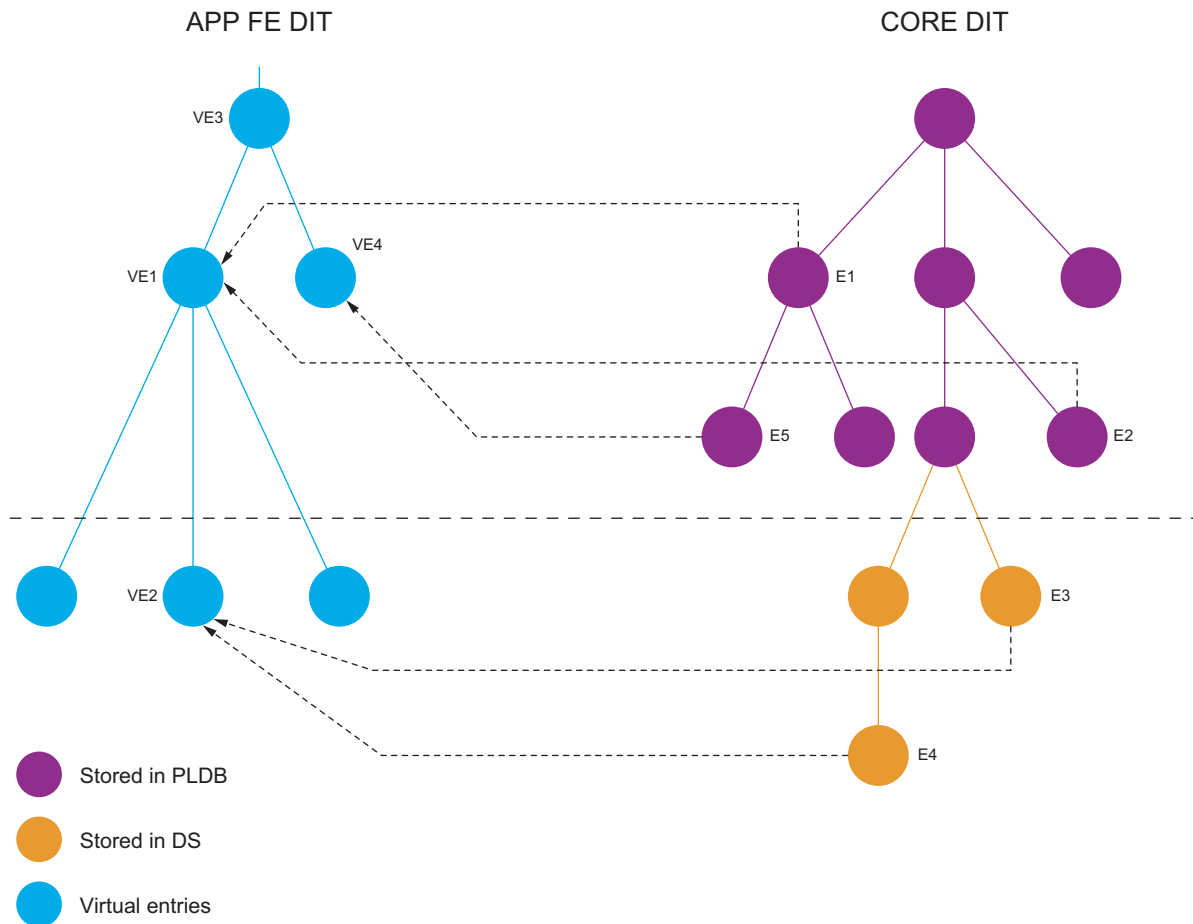


Figure 2 Mapping a Virtual Tree from Core DIT

Figure 2 shows an example of mapping. Four virtual entries (VE1, VE2, VE3, and VE4) are defined in the APP FE DIT virtual tree structure. Attributes used in mapping are taken from the core DIT tree. The attributes and object classes used in defining the virtual tree can be defined in the data view schema or in a CUDB schema.

Figure 3 explains the mapping definition of the first virtual entry, VE1. VE1 consists of the attributes of entries E1 and E2, which are defined in PLDB as part of the core DIT.

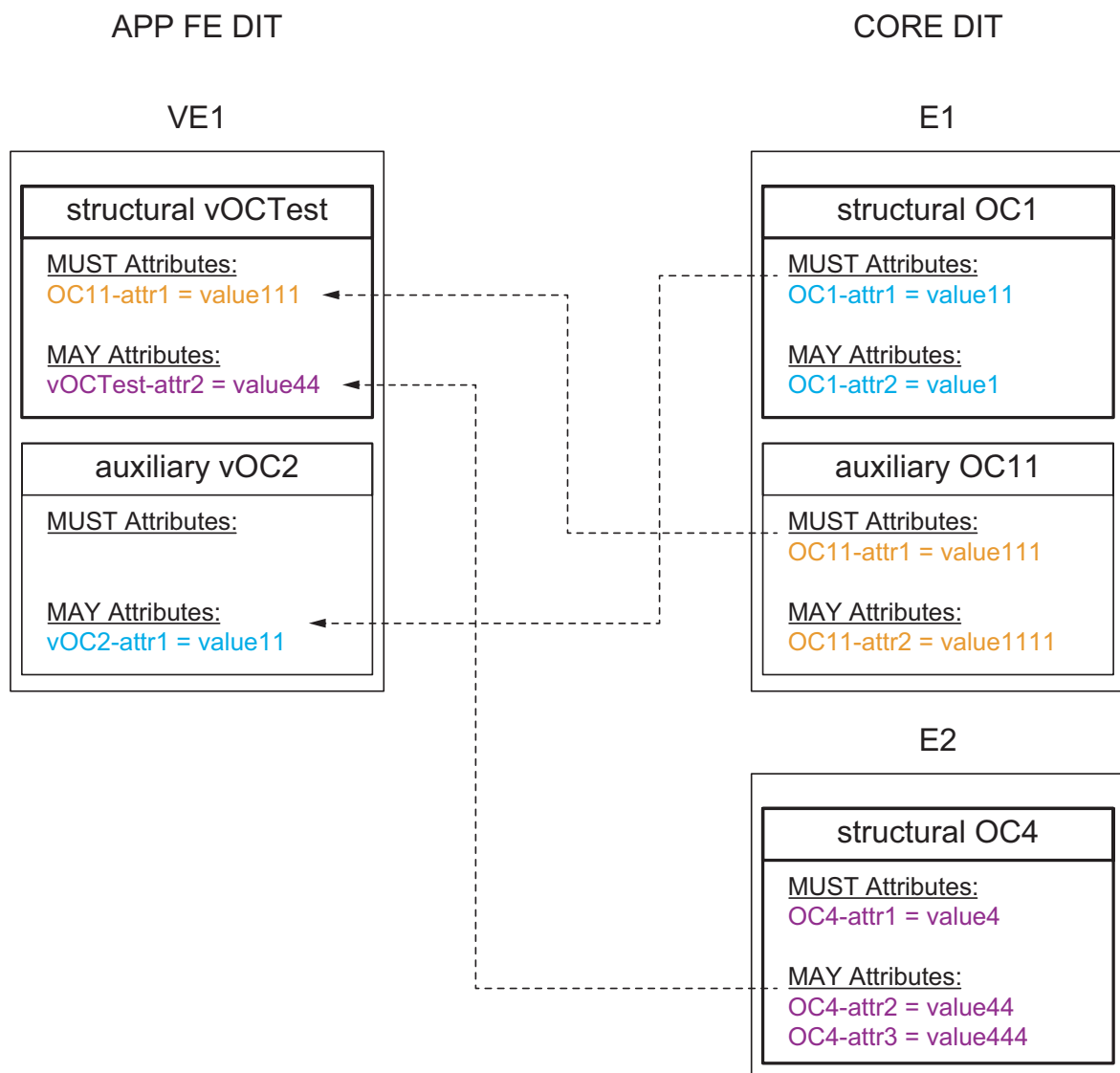


Figure 3 Mapping Definition of VE1

- Virtual attribute OC11-attr1 takes its value from the OC11-attr1 attribute of real entry E1 and uses the same attribute name. The attribute name is defined in a CUDB schema file.
- Virtual attribute vOCTest-attr2 is defined in a data view schema file and takes its value from the OC4-attr2 attribute of real entry E2.
- Virtual attribute vOC2-attr1 is defined in a data view schema file and takes its value from the OC1-attr1 attribute of real entry E1.

Figure 4 explains the mapping definition of the second virtual entry, VE2. VE2 consists of the attributes of entries E3 and E4, which are defined in a Data Store Unit Group (DSG) as part of the core DIT. Both entries are from the same DSG.

Note: Combining attributes from different DSGs is not allowed on Distribution Entry (DE) level.

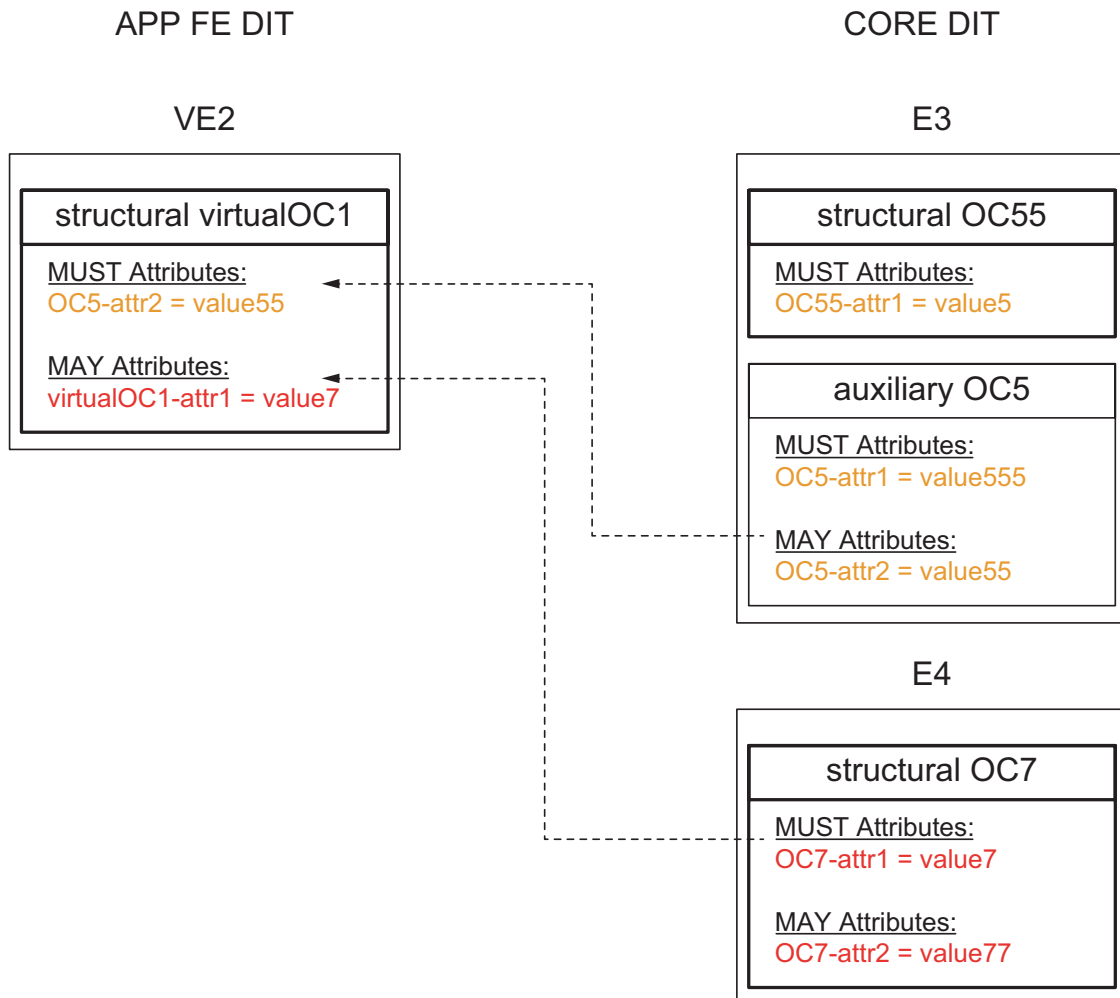


Figure 4 Mapping Definition of VE2

- Virtual attribute **OC5-attr2** takes its value from the **OC5-attr2** attribute of real entry **E3**, and uses the same attribute name. The attribute name is defined in a CUDB schema file.
- Virtual attribute **virtualOC1-attr1** is defined in a data view schema file and takes its value from the **OC7-attr2** attribute of real entry **E4**.

Virtual entry **VE3** is example of a scaffolding virtual entry. It has no content except for its naming attribute. For more details about scaffolding virtual entries, refer to Section 2.6 on page 10.

Virtual entry **VE4** is a one-to-one mapped entry. The value(s) of its attribute(s) and object class(es) are the same as those of the corresponding entry in the core DIT, entry **E5**. For more details about one-to-one mapped entries, refer to Section 2.7 on page 10.



2.5 Principles of LDAP Data Views

- A virtual LDAP tree must be defined from the root entry downwards.
- A virtual entry is identified by its DN in a virtual tree.
- A virtual entry must include at least one `STRUCTURAL` object class.
- A virtual entry may or may not include `AUXILIARY` object classes.
- All `MUST` attributes of an object class (`STRUCTURAL` or `AUXILIARY`) in a virtual entry must have a mapping rule describing from where in the core DIT (entry and attribute) to retrieve their value.
- `MAY` attributes of an object class (`STRUCTURAL` or `AUXILIARY`) in a virtual entry may or may not include a mapping rule describing from where in the core DIT (entry and attribute) to retrieve their value.
- A virtual entry must be defined in a way that the values of its attributes are retrieved from core DIT entries in either PLDB or DSG, but not from both.
- A virtual entry built using data from DSG must be built using data from a single DSG.
- Defining a virtual entry built using no CUDB data is possible. Such entries consist of a structural object class with a single `MUST` attribute containing the naming attribute of the entry. A scaffolding virtual entry is useful in providing structure and support to a virtual LDAP tree. For more details about scaffolding virtual entries, see Section 2.6 on page 10.
- A value of an attribute in a virtual entry, the target attribute, must get its value from an attribute in an entry from the core DIT, the source attribute, sharing the same `SYNTAX` and multiplicity (`SINGLE-VALUE` or `multivalued`). The `EQUALITY`, `SUBSTRING`, and `ORDERING` rules of the target and source attributes may be different.
- A source attribute must be identified by its name and the full DN of the core DIT entry where they are stored.
- The naming attribute of a virtual entry must be included in one of its structural or auxiliary object classes.
- The value of the naming attribute of a virtual entry gets its value from the DN of the virtual entry. The definition of the virtual entry cannot include a mapping rule to provide a value for the naming attribute.
- A virtual entry can be directly mapped to an entry in the core DIT, meaning that the virtual entry and the core DIT entry will contain the exact same `STRUCTURAL` and `AUXILIARY` object classes, and have the same naming attribute. Such entries are known as one-to-one mapped entries. For more details about one-to-one mapped entries, see Section 2.7 on page 10.

- A virtual tree is defined in a generic way. Consequently, multiple virtual entries can be mapped within one mapping rule, using variables. The two types of variables which could be used in a mapping definition are the following:
 - Regular variable: the value of the variable in the virtual entry is replaced by a value from a DN of the original LDAP request, see Figure 5.
 - Variable from alias in core DIT (also known as a `variableDefinition`): the value of the variable of the virtual entry is replaced with the alias contained in the core DIT entry.

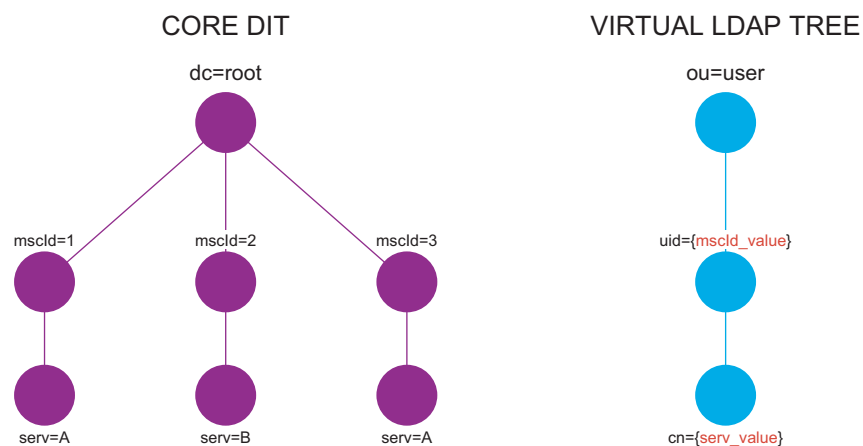


Figure 5 Regular Variables

2.6 Scaffolding Virtual Entries

Scaffolding virtual entries in a virtual view are entities in the virtual tree without any content or mapping information, except for their naming attribute. They are mainly used to construct parts of the virtual tree structure.

They can be defined in any part of the virtual tree.

They consist of a single, structural object class containing a single `MUST` attribute, the naming attribute of the virtual entry.

Scaffolding virtual entries belong neither to PLDB nor to DSG data. They only exist within a view and access to a database cluster is not required to build them.

Scaffolding virtual entries support the LDAP search operation only.



2.7 One-to-one Mapped Entries

Write operations through views are restricted in order to ensure the integrity of data and schema rules in the core DIT. However, one-to-one mapped entries exist as a dedicated type of entry that can be added, deleted, or have its content modified through views freely, with only the restrictions that apply for any real entry.

Adding or deleting application-specific entries cannot affect core DIT structure, so a one-to-one mapped entry can be used if an application needs to add its own data to a view.

A one-to-one mapped entry must be a virtual entry that is a clone of a corresponding entry in the core DIT. The virtual entry has the exact same contents as the core DIT entry and can be defined in any part of the virtual tree.

One-to-one mapped entries and their core DIT counterparts must have the same naming attribute. If an LDAP operation is performed on one-to-one entry whose naming attribute is different than the naming attribute of its original, return 19, `Wrong naming attribute`.

For more information on how to configure one-to-one mapping, refer to *CUDB LDAP Data Views Management*, Reference [2].





3 Description of Runtime Behavior

Access to CUDB data through a view is similar to direct access to the core DIT. The same LDAP requests are supported, but, because mapping between the core DIT and a virtual tree is flexible, restrictions and limitations are in place on certain LDAP operations, primarily on write operations such as add, delete, and modify, to ensure data and schema rules integrity.

These limitations make views ill-suited for provisioning operations. The guiding principle is that provisioning must be executed by an LDAP user accessing the core DIT, not by an LDAP user accessing CUDB through a view. Accordingly, an LDAP View cannot be assigned to provisioning or re-provisioning user, and an LDAP user assigned to a view cannot become provisioning or re-provisioning user. For more details about LDAP users and their configurations, refer to *CUDB Node Configuration Data Model Description*, Reference [3].

LDAP operations through views are atomic, meaning that they are either fully executed or not executed at all.

3.1 General Rules

- Virtual entries are not stored in a database, but built by the LDAP Data Views engine during runtime on an as-needed basis.
- A virtual entry can materialize in a view if retrieving the values for all **MUST** attributes of its **STRUCTURAL** object class and all **MUST** attributes of the object class containing its naming attribute is possible.
- An object class of an existing virtual entry can materialize in a view if retrieving a value for all of its **MUST** attributes is possible. **MAY** attributes in an object class can materialize if both of the following conditions are met:
 - a mapping rule to provide a value for them exist.
 - retrieving a value for them is possible.
- The `objectClass` attribute in a virtual entry is built during runtime and will only show the object classes which can appear.
- The views engine does not run parentage checks due to runtime costs. For core DIT entries, structure is explicit and enforced at entry-creation and deletion time. It is impossible to create a parentless entry and it is impossible to delete a non-leaf entry. Both conditions are resource-efficient to check.

Since virtual entries are built on the go, parent-child relationships are not explicit and must be computed. To verify the existence of virtual entry,



every other entry in the virtual tree above it must be tested as well during runtime therefore it is not done for LDAP views.

Note: The lack of parentage checks is not compliant with the LDAP RFCs.

Views show orphan (parentless) entries if they are requested by `baseName`, but they do not appear in a `wholeSubtree` search starting from one of their ancestors.

- Operations on one-to-one mapped entries are not subject to any of the limitations regular virtual entries suffer, but they are subject to all the limitations that apply to core DIT entries.

3.2 Search Requests

LDAP search operations through views are mostly unrestricted. Existing restrictions are related to crossing PLDB-DSG boundaries.

Search operation is explained through Figure 6.

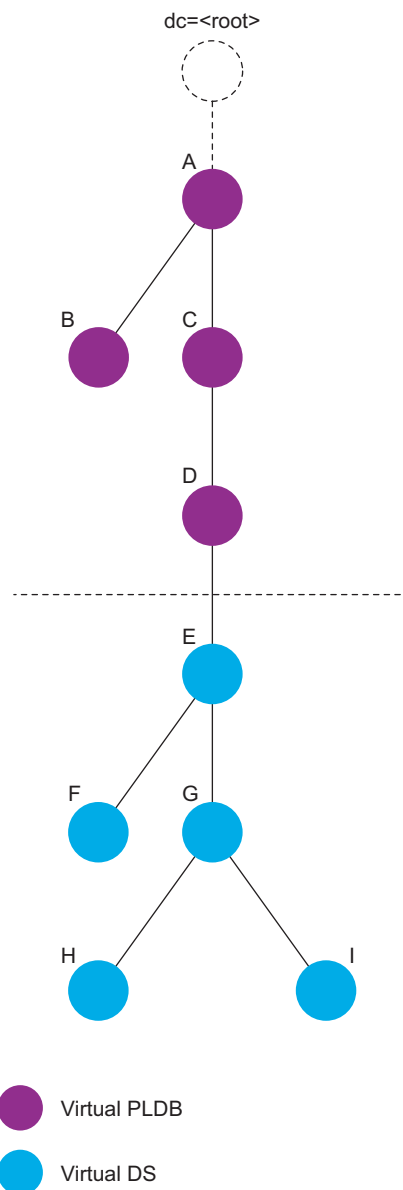


Figure 6 Simplified Tree Structure

The LDAP search operation follows the rules below:

- If **baseObject** is a virtual entry (**entry A**) that is mapped to PLDB data, and the **scope** is **wholeSubtree**, return all connected entries in the subtree that are built from PLDB data (entries **A**, **C**, **D**) and entry **B**, a scaffolding virtual entry.
 - If there are any DSG virtual entries in the subtree, return 53, Full tree searches are not allowed for views.



- If **baseObject** is a virtual entry (**entry A**) that is mapped to PLDB data, and the **scope** is **baseObject** or **singleLevel**, return the corresponding virtual entries from PLDB, (**entry A** for **scope baseObject**; entries **B**, **C** for **scope singleLevel**).
- If **baseObject** is a virtual entry (**entry D**) that is the lowest entry in the tree from PLDB and has children from DSG, the following rules apply:
 - If the **scope** is **baseObject**, return the virtual PLDB entry (**entry D**).
 - If the **scope** is **singleLevel**, return the first level of DSG virtual entries under the base virtual entry (**entry E**).
 - If the **scope** is **children**, return all child entries from DSG under the base virtual entry (entries **E**, **F**, **G**, **H**, **I**).
 - If the **scope** is **wholeSubtree**, return PLDB entry (**entry D**) and 53, Full tree searches are not allowed for views.
- If the **baseObject** is a virtual entry that is in DSG (**entry E**):
 - If the **scope** is **baseObject**, build a virtual entry for the base object (**entry E**).
 - If the **scope** is **singleLevel**, build the first level of virtual entries under the **baseObject**, and return it (entries **F**, **G**).
 - If the **scope** is **wholeSubtree**, build all the virtual child entries of the view under the base object (entries **E**, **F**, **G**, **H**, **I**).

Virtual aliases point to virtual entries whose DN is defined in **aliasedObjectName** of the entry in the core DIT. The virtual **aliasedObjectName** attribute must contain a value that is a valid virtual DN, requiring extra data to be added to the core DIT. See Figure 7 for an example of mapping an alias: virtual entry **VE1** is mapped to entry **E3**, and it points to virtual entry **VE2**.

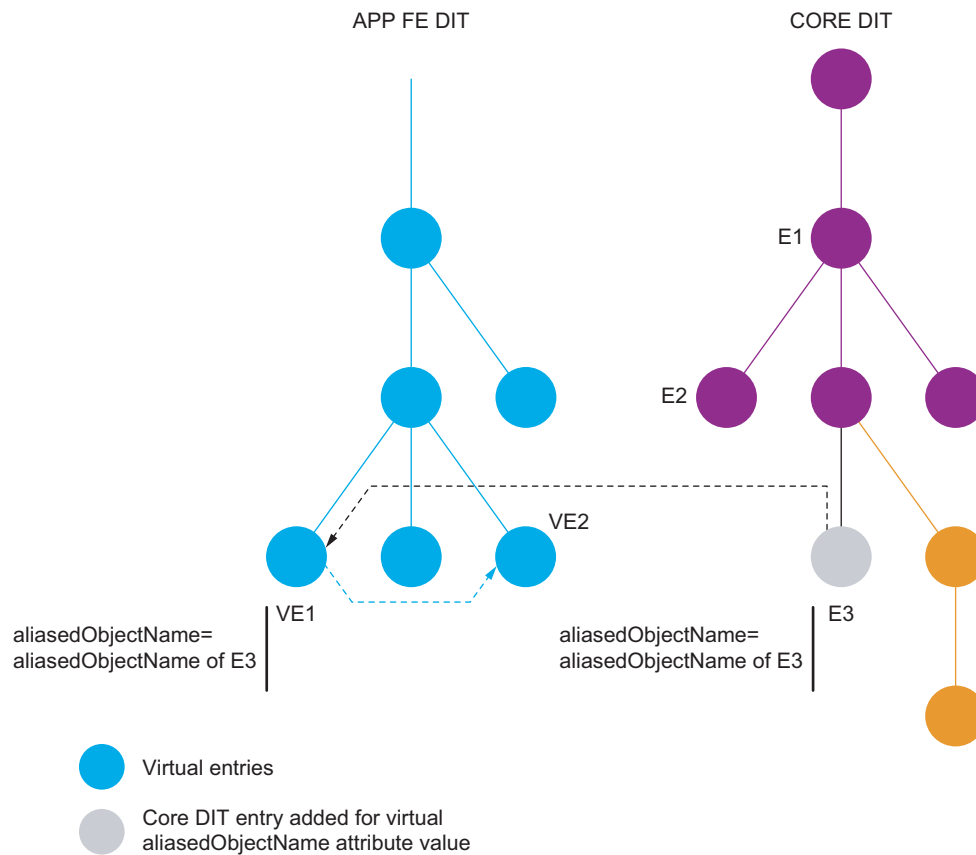


Figure 7 Alias Mapping

3.3 Write Requests

The flexibility allowed in defining virtual entries requires limitations to ensure the integrity of data stored in the core DIT.

Core DIT write operations only enforce the data integrity of the core DIT. Views write operations enforce integrity of the data in the view and the data in the core DIT, but not within other views.

Modifying data in a view will result in the data modified in the core DIT as well. Any change in the core DIT (directly or through a view) affects each view using the relevant entries.



Example

If a `MAY` attribute of a virtual entry in view A is deleted, the mapped `MAY` attribute of the corresponding core DIT entry is also deleted. Consequently, if the core DIT attribute was the source of a `MUST` attribute in an entry of view B, the affected virtual entry will no longer appear in either a `singleLevel` or a `wholeSubtree` search and its child entries will no longer appear a `wholeSubtree` search.

As virtual LDAP entries may be built using data from multiple core DIT LDAP entries, a single write operation through views affecting multiple attributes will affect multiple core DIT entries. This is a side effect of using the LDAP Data Views function, as standard LDAP write requests, excepting LDAP transactions, can only act on a single LDAP entry.

Note: It is not allowed to write data hosted in PLDB entries through operations done over virtual entries that exist on the DSG level, even if the virtual entries can be built from attributes or objects that exist in associated PLDB entries (for example, by means of using DSG to PLDB variable definitions).

LDAP modify operations performed over those virtual entries will be rejected with error code 53: Operation not supported in LDAP views.

3.3.1 Add

Except for one-to-one mapped entries, add requests through a view are forbidden. As adding a one-to-one mapped entry means creating an entry in the core DIT, parentage checks are run. If a parentage check fails, the add request will fail as well.

If an LDAP add request arrives from an LDAP user with a view assigned and the entry is not a one-to-one mapped entry, return 53, `ldapadd` operation is not supported in `ldap` views.

3.3.2 Delete

Except for one-to-one entries, delete requests through a view are forbidden. As deleting a one-to-one mapped entry means removing an entry from the core DIT, leaf-entry checks are run. If a leaf-entry check fails, the delete request will fail as well.

If an LDAP delete request arrives from an LDAP user with a view assigned and the entry is not a one-to-one mapped entry, return 53, `ldapdelete` operation is not supported in `ldap` views.



3.3.3 Modify Requests

Modify request through views are allowed as long as they do not alter the structure of entries in the core DIT. Any modify request that would result in either of the following will be rejected:

- Object classes being added to or deleted from core DIT entries.
- Entries in the core DIT being added or deleted.

It is possible to add values to the `objectClass` attribute of a virtual entry, but the write operation will not persist unless the LDAP request includes all the required input to create a valid object class.

Example

If a new object class X consisting of only `MAY` attributes is added to virtual entry Y and the request does not include a value for each of those `MAY` attributes in the same LDAP request, the write operation will not persist. The next time virtual entry Y appears, object class X will not be listed in its `objectClass` attribute.

- When an LDAP modify operation is executed on an auxiliary object class or an attribute which is not defined in a mapping file return 16, `Attribute does not exist in the explicit mapping`.
- When modifying an attribute to a value higher than the maximum allowed in the core DIT, return 21, `Error in core DIT`.

3.3.3.1 Modify-Add

The following rules apply to an LDAP modify (modify-add) request operation:

- When adding a virtual attribute implies adding a `MUST` attribute to a structural object class in the core DIT, return 53, `Operations which imply adding object classes to core DIT entries are not allowed in views`.
- When adding a virtual attribute implies adding a `MUST` attribute to an auxiliary object class in the core DIT, return 53, `Error in core DIT`.

3.3.3.2 Modify-Delete

The following rules apply to an LDAP modify (modify-delete) request operation:

- When deleting an attribute of a virtual entry requires deleting a `MUST` attribute of a structural or auxiliary object class in the core DIT, return 53, `Error in core DIT`.





4 Dependencies and Interactions

An LDAP user using a view does not interfere with other LDAP users accessing data through other views or the core DIT. However, the following restrictions of the LDAP Data Views function must be taken into account:

- Data import and export procedures can only be performed through the core DIT.
- An LDAP user cannot use an LDAP Data view and the core DIT at the same time. Once a view is assigned to a user, the user can only access data through the assigned view.
- Simple Object Access Protocol (SOAP) Notifications are configured on the core DIT attributes, object classes, and tree structure. It is impossible to configure notifications involving virtual entries other than indirectly. Notifications can be configured for the real entries out of which the virtual entries are built.
- Only application data stored in the PLDB and DSG is available through a view. Application counters exposed through the LDAP interface or internal LDAP server configuration are unavailable.
- UDC application FEs Collision Detection Control (CDC) mechanism can be used in LDAP data views with the limitation that the virtual entry hosting the CDC attribute is built by mapping attributes from one, and only one, real entry.

In this case, application FEs can rely on the CDC attribute value to control interwoven concurrent read and writes operations over the same piece of data, regardless if they are accessing through the view or the core DIT.

- Use of the `createTimestamp` and `modifyTimestamp` attributes is not allowed in the mappings of virtual entries, except for one-to-one mapped entries.





5 Operation and Maintenance

5.1 Configuration

The LDAP Data View function is configured through Information Model Management (IMM). A set of IMM classes and attributes are defined to provide the configuration options of this function.

For more details about LDAP Data Views IMM classes and attributes, refer to *CUDB Node Configuration Data Model Description*, Reference [3].

Before creating an LDAP View through IMM, the configuration files, the mapping configuration, and the data view schema file must be stored at the proper location.

Once a view has been configured, it must be assigned to an LDAP User through IMM in order to be used.

For more details about the configuration of the LDAP Data Views function, refer to *CUDB LDAP Data Views Management*, Reference [2].

Note: The LDAP Data Views function can only be used if the Application Facilitator Value Package is available.

5.2 Fault Management

Not applicable.

5.3 Performance Management

Not applicable.

5.4 Security

5.4.1 Access Control List (ACL)

Access Control List rules are fully supported by the LDAP Data Views function and can be configured in the same way as for the core DIT by modifying the `acls.conf` file. For more information about configuring access control for the LDAP Data Views function, refer to *CUDB Security and Privacy Management*, Reference [4].



5.5

Logging

Not applicable.



Glossary

For the terms, definitions, acronyms, and abbreviations used in this document, refer to *CUDB Glossary of Terms and Acronyms*, Reference [5].





Reference List

Ericsson Documents

- [1] *CUDB Node Schema Update*
- [2] *CUDB LDAP Data Views Management*
- [3] *CUDB Node Configuration Data Model Description*
- [4] *CUDB Security and Privacy Management*
- [5] *CUDB Glossary of Terms and Acronyms*