ERICSSON

CUDB Application Integration Guide

USER GUIDE

7/1553-HDA 104 03/10-V1 Uen A

Contents

1 Introduction

This document provides information required by application Front Ends (FEs) to integrate with the Ericsson Centralized User Database (CUDB).

1.1 Document Purpose and Scope

This document provides the information required by applications FEs to integrate in CUDB and information regarding the services offered by CUDB to applications.

The intended audience for this document is network solution architects, application product managers and application engineers in charge of carrying out the integration of a specific application with CUDB.

1.2 Revision Information

Rev. A

This document is based on 7/1553-HDA 104 03/9 and contains the following changes:

- Content rearrangement throughout the document.
- Section 3 on page 5: Updated LDAP Data Views information.
- Section 4.5 on page 8: Updated description of PLDB and the logic of accessing PLDB replicas by application FEs.
- Section 4.7 on page 11: Updated with disk storage system information.
- Section 9 on page 23: Added new section on Electronic SW Licenses.

1.3 Typographic Conventions

Typographic conventions can be found in the following document:

- *Typographic Conventions*

2 Preconditions

A general knowledge of CUDB is required to fully understand the concepts presented in this document. For general information on CUDB, refer to *CUDB Technical Product Description*, Reference [1].

3 Overview

CUDB is a highly available, geographically redundant, real-time, distributed Subscriber-Centric Database exposed as a Lightweight Directory Access Protocol (LDAP) directory to applications (such as HLR, AuC, and HSS). CUDB is the central entity for data storage in Ericsson's offering providing the data layered architecture, that separates applications logic and data storage. With this architecture, separate dimensioning of processing and storing resources can be achieved by decoupling both logically and physically application/business logic and the subscriber data. Furthermore, it allows the unification of applications (such as HLR, AuC, HSS-EPC, and HSS-IMS) specific data under a common subscriber profile which is stored in a common database. With this architecture, application FEs, or applications from here on, hold the application business logic while CUDB provides the common database supporting the storage of the common subscriber profile for applications. For more details on CUDB functions, refer to *CUDB Technical Product Description*, Reference [1].

CUDB provides a basic LDAP Directory Information Tree (DIT) extended by applications to store and query their data. The most basic information an application must provide to be integrated with the CUDB system is the data to be stored and managed by the application. The data is defined in terms of LDAP schemas that contain the types of information stored in CUDB. Due to the distributed nature of the CUDB database system, it is important for the application to understand how data are distributed and replicated in CUDB to organize its data more efficiently.

To optimize the storage space of CUDB, applications must take into consideration certain aspects of the way CUDB maps LDAP schemas into its internal structures.

CUDB provides mechanisms to optimize massive LDAP search queries by means of index definition.

Access to the data by applications using the CUDB LDAP interface requires the configuration of LDAP users, typically one per application, to be able to provide different types of access to different applications.

The LDAP Data Views function supports accessing stored data through customizable views. The function makes it possible for applications to access the data stored in CUDB through a custom tree structure (that is, a custom DIT) and a custom schema. Several LDAP data views can be defined to accommodate different kinds of application FEs.

LDAP users can be configured to either access CUDB using the ‘‘native’’ view or core DIT or to use one of the defined LDAP data views.

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

CUDB offers the possibility of defining custom notifications that may be sent to application FEs through the Simple Object Access Protocol (SOAP) protocol. The data that would fire the custom notification and the set of extra conditions required for the notification to be sent together with the data to be included in the notification message are configurable.

CUDB also provides applications with the possibility of defining their own set of counters related to the data stored.

All these possibilities and the information to be provided by the applications are outlined in the following sections.

4 Data Management

This section contains details about the data management. For more information, refer to *CUDB LDAP Data Access*, Reference [2].

4.1 LDAP Views

CUDB maintains views to support flexible data organization. If an LDAP user has an LDAP view assigned to it, then it can access the data through that view. Each view has its LDAP schema loaded into the CUDB, where entries are composed by mapping attributes from the default view, resulting in the building of a custom DIT. Each view has a mapping file, which describes how the view entries are composed from the attributes contained by the default view.

The LDAP Data Views function allows access to the data in different data structures through the LDAP interface.

All the attributes that are reachable through views should be already defined in the default view.

Once a user with an assigned view establishes a new LDAP bind, all the following LDAP operations are performed against the LDAP view selected for this user. LDAP data views are not available for export and import procedures, or for notifications. Notifications must be configured according to core DIT distinguished names and attributes, and data in SOAP messages is, likewise, identified by its core DIT distinguished names and attributes. Users having no view assigned will see the PL and DS data through the default LDAP tree. For users that have a view set through configuration, the given LDAP schema of that view will apply.

For more information on LDAP views, refer to *CUDB LDAP Data Views*, Reference [3].

4.2 LDAP DIT

CUDB is exposed as an LDAP DIT to applications. CUDB provides a basic LDAP DIT to be extended by applications to store and query their data.

Normally, applications place their data under the predefined CUDB branches, but they can also create their own branches under the DIT root entry. For more details on the CUDB LDAP basic DIT, refer to *CUDB LDAP Data Access*, Reference [2] and *CUDB LDAP Interwork Description*, Reference [4].

4.3 LDAP Schemas

CUDB provides a typical LDAP *core* schema and a CUDB-specific LDAP schema with object classes and attributes that are used in the predefined entries that build the LDAP basic DIT. These attributes and the object classes can also be used by applications for their own entries.

Applications can also provide their own schema files that define specific object classes and attributes to be used by the applications data. Schemas containing object classes and attributes common to several applications are also supported, provided that object classes and attribute definitions are not repeated in different LDAP schemas.

CUDB provides tools that automatically and transparently perform a translation between LDAP schema elements and the internal database structures used by CUDB to support the LDAP schemas required. The schemas provided by applications must comply with some requirements to be usable by these tools. Refer to *CUDB Node Schema Update*, Reference [5] for more details.

LDAP schemas are typically provided during CUDB system installation, but new LDAP schemas can also be included once the system is up and running. With certain limitations, CUDB also offers the option of updating existing LDAP schemas, once an application has been integrated in CUDB. For more details on schema update, refer to *CUDB Node Schema Update*, Reference [5].

For more information about LDAP schemas, refer to *CUDB LDAP Data Access*, Reference [2].

4.4 Identities

Identities are special types of entries in CUDB that are applied to locate subscribers using network identifiers (such as MSISDN, IMSI, and so on) by means of the LDAP alias mechanism. Refer to *CUDB LDAP Interwork Description*, Reference [4] for more details about identities.

Identities must be defined at installation or schema update time. A container for each type of identity is created automatically under a CUDB specific branch at installation and schema update time under a CUDB-specific branch. As identities are normally shared by different applications, it is required that a common LDAP schema containing the attributes and object classes used by identities is provided for all applications. Refer to *CUDB Node Schema Update*, Reference [5] for more details about identities definition.

For more information about identities, refer to *CUDB LDAP Data Access*, Reference [2].

4.5 Data Distribution

CUDB contains two types of storage layers in which applications place their data: the Processing Layer Database (PLDB) and the Data Store Unit Groups (DSGs).

- The PLDB is an indexing database replicated in certain nodes of all CUDB sites for the data stored in the DSGs. It also stores the complete set of subscriber identities. The PLDB may as well be used to store certain data which does not have massive size (for example common subscriber profiles) and is not related to one specific subscriber.

  Note: The PLDB does not offer the level of scalability provided by the DSGs. Therefore, it is recommended to study carefully what is stored in the PLDB, so that its capacity is not exceeded in a specific deployment.

- The DSGs are partitions of the subscribers database. Each DSG holds the data for a group of subscribers. Subscribers are distributed at provisioning time in the DSGs available in the system. Depending on the level of redundancy of the system one or more replicas of each DSG, called Data Store (DS) Units are present in the CUDB system. Each DS is assigned to a CUDB node.

  Note: To retrieve the data for a specific subscriber, a hop may be needed from the CUDB node where the request arrived to the node that contains the DS master replica where the data are stored.

Data that require extra performance in retrieval and is not massive in size is a good candidate to store in the PLDB. This is because the application FEs always access to a node that hosts a PLDB replica, and therefore no hops to other CUDB nodes are required when accessing PLDB data.

CUDB determines automatically if a certain LDAP entry is stored in the PLDB or in DSGs based on the position the entry has in the LDAP DIT. For details on which LDAP branches are stored in the PLDB and which LDAP branches are stored in DSGs, refer to *CUDB LDAP Interwork Description*, Reference [4].

For more information on how data is distributed in CUDB, refer to *CUDB LDAP Data Access*, Reference [2].

4.6 Storage Space Considerations

As stated previously, CUDB provides tools that automatically and transparently perform a translation between LDAP schema elements and the internal database structures used by CUDB to support the LDAP schemas required. Refer to *CUDB Node Schema Update*, Reference [5] for further details.

For optimizing the storage space, the following information is provided regarding the mapping performed by CUDB from LDAP types into internal database types:

- For each entry in the LDAP DIT (of type other than identity) CUDB holds a row in a master table that contains the Distinguished Name (DN) of the entry among other data. Therefore, the longer the DN, the more space is taken by the entry in the database.

- For each identity entry, CUDB holds a row in a table in the PLDB for each identity type holding the values for the identity attributes (but not the complete DN).

- For each object class defined in any schema file, CUDB uses one table that holds one row for each LDAP entry containing that object class. If none of the attributes of the object class has a value for the entry, no row is present in the object class table. Therefore, it is recommended, to optimize the space occupied by data, to group attributes used together by the application (when one of these attributes is present all of them in the object class are present) in the same object classes.

- Each object class table contains a key column for each row and one column for each single-attribute belonging to the object class. Multi-value attributes are stored in separate tables (one for each pair of object classes or attributes) and CUDB uses one row of these latter tables for each multi-value attribute value.

The mapping of LDAP attribute types into internal database types performed by CUDB tools is shown in Table 1.

Table 1 Mapping of LDAP Attribute Types into Internal Database Types Performed by CUDB

LDAP attribute type	**Internal Database type**
Bit String	`BIT(LEN)`
Boolean	`VARCHAR`
Country String	`CHAR(2)`
Directory String	`VARCHAR CHARACTER SET UTF8`(1)
GeneralizedTime	`VARCHAR`
NumericString	`VARCHAR`
DN	`VARCHAR(512) CHARACTER SET UTF8`
Integer	`TINYINT/SMALLINT/MEDIUMINT/ BIGINT/INT`
OctetString	`VARBINARY`
OctetString{M} where M<255	`VARBINARY(255)`
OctetString{M} where M <= 4 KB	`VARBINARY(M)`
OctetString{M} where 4 KB < M <= 64 KB	`BLOB(M)`

LDAP attribute type	Internal Database type
OctetString{M} where 64 KB < M <= 16 MB	MEDIUMBLOB(M)
OctetString{M} where 16 MB < M <= 4 GB	LONGBLOB(M)
Other	VARCHAR

(1) A UTF-8 encoding of a Unicode character set uses one to three bytes per character.

4.7 BLOB Configuration and Storage Options

CUDB offers the possibility of storing Binary Large Objects (BLOBs). Attributes of BLOB type are defined at installation time together with the storage support option, Random Access Memory (RAM) or the disk storage system, for them on a per LDAP attribute basis. Refer to *CUDB Binary Large Object Attributes Management*, Reference [6] for more details on how BLOBs are configured.

The decision on whether a certain BLOB attribute is to be kept on the disk storage system or in RAM must take into consideration both performance and storage capacity requirements. BLOB attributes on the disk storage system have higher access time than BLOBs in memory, but leave more main memory available for other data to be stored.

5 LDAP Massive Search Indexes

For information about LDAP massive search, refer to *CUDB LDAP Data Access*, Reference [2].

5.1 Candidate Queries for Search Index Optimization

Search indexes are used by CUDB when an LDAP search is received with a filter that includes one or more indexed attributes, and certain conditions are fulfilled by the filter. Search indexes are used by CUDB for optimizing queries, so that entries complying with a specific filter are selected without the need of retrieving all the possible candidates (according to the base DN and scope parameters of the search) from the database to check if the filter is fulfilled.

Note: The optimizations with indexes are only applicable for one level and subtree search operations.

Search indexes are not applicable to identity entries, even if these entries contain an indexed attribute.

Consider that LDAP queries that specify `search` or `always` alias dereferencing are not optimized as much as queries not specifying these options. This is because alias dereferencing in this fashion always requires extra search operations that cannot take advantage of the search indexes.

A query is optimized by search indexes if certain conditions are fulfilled. These conditions are defined in condition sets, and are described in Section 5.1.1 on page 13 and Section 5.1.2 on page 15. To optimize a query by search indexes, at least one of these condition sets must be met (both condition sets are recursive).

5.1.1 Condition Set 1

The first condition set is as follows:

- The filter must be an *&* ('and') filter, and there must be at least one term in the 'and' expression that fulfills (1).
- The filter must be simple, and must be one of these types: *equality, approx, greater or equal, less or equal*. Also, the attribute in the filter must be indexed.

Consider the order in which indexed attributes appear in a filter. If more than one indexed attributes are present in the filter that fulfill the second condition, then try to prioritize the one providing the most restrictive set of entries before

the rest to execute the query more efficiently by CUDB. Some examples for query optimization are listed below.

Example 1

Consider that `subsId` and `CSP` are indexed attributes, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is as follows:
  ```
  (&(subsId=1)(CSP=2)(&(CSPVERS=3)\
  (!(TIF=3))(objectClass=CsPsCAMELProf))).
  ```

This search fulfills Condition Set 1, so the `(subsId=1)`, `(CSP=2)` and `(objectClass=CsPsCAMELProf)` conditions are used to optimize the search.

However, if, for example, the number of entries fulfilling `(subsId=1)` is 10000, and the number of entries fulfilling `(CSP=2)` is 2, it is recommended to rewrite the filter as below to execute it more efficiently:

```
(&(CSP=2)(subsId=1)(&(CSPVERS=3)(!(TIF=3))\
(objectClass=CsPsCAMELProf)))
```

Example 2

Consider that `subsId`, `CSPVERS` and `CSP` are indexed attributes, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is as follows:
  ```
  (&(subsId=*)(CSP=2)(&(CSPVERS=3)\
  (!(TIF=3))(objectClass=CsPsCAMELProf))).
  ```

This search fulfills Condition Set 1. Therefore, the index optimization is based on `(CSP=2)`, but the `(subsId=*)`, `(CSPVERS=3)` and `(objectClass=CsPsCAMELProf)` conditions are also used to filter the search request before the data for the candidate entries is retrieved.

Example 3

Consider that `subsId` and `CSP` are indexed, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is `(&(subsId>=1)(subsid<=3))`.

This search fulfills Condition Set 1. Therefore, the search is optimized by filtering by the `(subsId>=1)` and `(subsid<=3)` conditions in the optimization

phase before the complete set of candidate entries is retrieved from the database.

5.1.2 Condition Set 2

The elements of the second condition set are listed below. In this case, at least one of the conditions must be fulfilled for optimization.

- The filter must be a / ('or') filter, and all the terms in the 'or' must fulfill (2).
- The filter must be an *&* ('and') filter, and at least one term in the 'and' expression must fulfill (2).
- The filter must be a simple one of these types:
  equality, approx, presence, substring, greater or equal, less or equal.
  The attribute in the filter must be indexed.
- The filter must be a simple one of these types: *equality, approx,*. The attribute in the filter must be an object class.

Some examples for query optimization based on this condition set are listed below.

Example 4

Consider that `subsId` and `CSP` are indexed attributes, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is as follows:
  `(&(|(subsId=1)(subsId=3))(objectClass=CUDBSubscriber))`.

This search fulfills Condition Set 2. In this case, the search is optimized by filtering the `(|(subsId=1)(subsId=3))` conditions in the optimization phase before the complete set of candidate entries is retrieved from the database.

Example 5

Consider that `subsId` and `CSP` are indexed attributes, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is as follows:
  `(subsId=*3*)`.

This search fulfills Condition Set 2. In this case, the search is optimized by filtering the `(subsId=*3*)` condition in the optimization phase before the complete set of candidate entries is retrieved from the database.

Example 6

Consider that `subsId` and `CSP` are indexed attributes, and an LDAP search query is received with the following properties:

- The `scope` is `wholesubtree` or `onelevel`.
- The `filter` is as follows:
  `(|(TIF=3)(CSP=2))`.

This search does not fulfill any of the condition sets (neither 1, nor 2). Therefore, this search is not optimized by search indexes.

Note: Queries fulfilling Condition Set 2 provide smaller gain as the queries fulfilling Condition Set 1. However, they still provide better performance compared to queries not optimized by search indexes.

If none of the Conditions Sets are fulfilled, the search is not impacted by the indexed attribute optimization.

5.2 Configuration

To configure Massive Search Indexes, refer to *CUDB System Administrator Guide*, Reference [7].

Note: These indexes must be defined before any entry with the indexed attributes is populated in CUDB. Otherwise, search operations may not work correctly.

5.3 Storage Space Considerations

Adding massive search indexes has an impact in the space taken by each LDAP entry in CUDB. For each massive search index a new column is added to the CUDB table that contains the DNs for each entry. If the entry has a value for the indexed attribute, the value is stored in this table and in the table that holds all the attributes of the entry. Attributes with value take double space if they are indexed. Depending on the type of the indexed attribute, a small overhead may be taken to hold the length, even if the entry does not have a value for the specific attribute.

6 LDAP Users and Access Control

To enable the applications to access the data stored in CUDB through the LDAP interface, one or several LDAP users must be defined with the corresponding credentials, configuration parameters, and Access Control Rules (ACR). Application FEs may bind to the CUDB LDAP interface with one of these users, then send the corresponding LDAP operations to CUDB. User groups can optionally be defined to share Access Control Rules between LDAP users.

For more details on LDAP User Management, refer to *CUDB LDAP Data Access*, Reference [2] and *CUDB System Administrator Guide*, Reference [7].

7 Notifications

CUDB offers the possibility of defining custom notifications to be sent to application servers when changes are detected in the data stored in the DSG. The notifications protocol is SOAP-based. For information on the specific notifications protocol, refer to *CUDB SOAP Interwork Description*, Reference [8].

CUDB allows the configuration of which LDAP attributes are monitored to send notifications in case they change. CUDB also allows the specification of a condition set on other related data to decide if the notification must be sent. The specific data to be sent in the notification (which could be the monitored data and/or any other data) can also be configured. For information on how to configure notifications in CUDB, refer to *CUDB Node Configuration Data Model Description*, Reference [9].

For more information on this feature, refer to *CUDB Notifications*, Reference [10]. In case the LDAP Data Views function is activated, see Section 4.1 on page 7 for more information about notifications related to LDAP views.

8 Application Counters

CUDB provides a framework that allows applications to define their own set of counters to be generated by CUDB. The framework is based on the definition of scripts that calculate the desired counters by accessing the data stored in CUDB and output them in XML files.

For details on how to configure these counters, refer to *CUDB Application Counters*, Reference [11].

9 Licensing

Licensing follows the CUDB commercial model where generic profile capacity licenses are defined;these generic licenses are meant to be used (extended) in case new applications are integrated with CUDB.

The description of the needed adaptations to cope with the integration of a new application from CUDB point of view is for Ericsson personnel.

Glossary

For the terms, definitions, acronyms and abbreviations used in this document, refer to *CUDB Glossary of Terms and Acronyms*, Reference [12].

Reference List

CUDB Documents

[1] *CUDB Technical Product Description*

[2] *CUDB LDAP Data Access*

[3] *CUDB LDAP Data Views*

[4] *CUDB LDAP Interwork Description*

[5] *CUDB Node Schema Update*

[6] *CUDB Binary Large Object Attributes Management*

[7] *CUDB System Administrator Guide*

[8] *CUDB SOAP Interwork Description*

[9] *CUDB Node Configuration Data Model Description*

[10] *CUDB Notifications*

[11] *CUDB Application Counters*

[12] *CUDB Glossary of Terms and Acronyms*