

CUDB Notifications

FACILITY DESCRIPTION

Copyright

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

Disclaimer

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

Trademark List

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



Contents

1	Introduction	1
1.1	Document Purpose and Scope	1
1.2	Revision Information	1
1.3	Target Groups	1
1.4	Prerequisites	2
1.5	Typographic Conventions	2
2	Overview	3
2.1	Prerequisites	3
2.2	Architecture	3
2.3	Description	4
2.4	Dependencies and Interactions	5
3	Operation and Maintenance	9
3.1	Activation	9
3.2	Configuration	9
3.3	Fault Management	19
3.4	Performance Management	19
3.5	Security	19
3.6	Logging	19
	Glossary	21
	Reference List	23





1 Introduction

This document provides a description of the optional notifications feature in the Ericsson Centralized User Database (CUDB).

1.1 Document Purpose and Scope

The purpose of this document is to describe the optional notifications feature from a functional and operational point of view.

1.2 Revision Information

Rev. A This document is based on 9/15534-HDA10403/9 with the following changes:

- Updated hardware information.

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

- Section 2.2 on page 3: Added information on sets of DEs.
- Section 2.3 on page 4: Added information on regular expression pattern.
- Section 3.2.2 on page 10: Updated information on `CudbNotificationObjectClass` and the `monitor` type.
- Section 3.2.3 on page 12: Added new section with examples of configuring new CUDB Notification Events using regular expressions.

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

- Section 3.2.2 on page 10: Updated to reflect support of extended POSIX regular expressions.
- Section 3.2.3.2 on page 16: Updated to reflect support of POSIX regular expressions.

1.3 Target Groups

As this document provides overall information about the notifications feature, it is intended for system developers, testers, administrators, or operators.



1.4 Prerequisites

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

1.5 Typographic Conventions

Typographic conventions can be found in the following document:

- *Typographic Conventions*



2 Overview

The notifications feature is used to inform Front End (FE) applications about modifications in data they store in a CUDB system.

2.1 Prerequisites

This section is not applicable to this feature.

2.2 Architecture

The notifications feature works on a Distribution Entry (DE) basis to provide information about changes in the attribute data stored in entries below the DEs in the Lightweight Directory Access Protocol (LDAP) tree. In other words, it reports modifications of data stored in the Data Store (DS) layer of a certain DE or set of DEs.

Figure 1 shows the notifications module, its components, and relationships with other CUDB modules and external FEs.

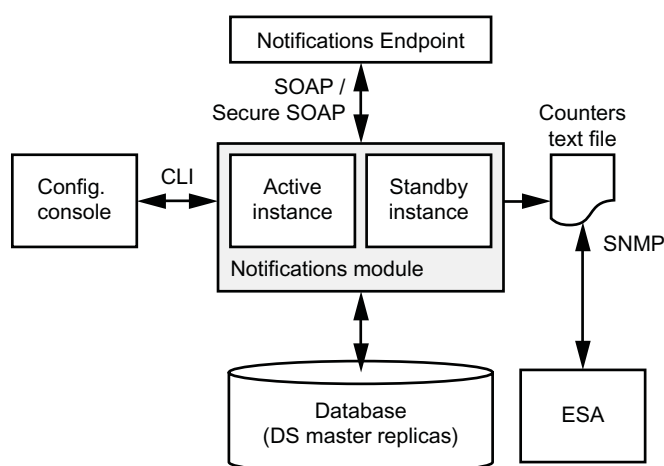


Figure 1 Notifications Module

The notifications module subscribes to the database events to monitor attribute changes, check for certain attribute values, and retrieve attribute contents to build the notifications.



Notifications to applications are sent as Simple Object Access Protocol (SOAP) messages over Hypertext Transfer Protocol (HTTP). CUDB acts as the SOAP client and the applications FEs act as the SOAP servers (referred to as endpoints from now on). For more information on the CUDB SOAP interface, refer to *CUDB SOAP Interwork Description*, Reference [1].

Use of Transport Layer Security (TLS) key and certificate is also possible in the SOAP interface. For more information on security in CUDB, refer to *CUDB Security and Privacy Management*, Reference [2].

The notifications feature has the following configuration options:

- It is possible to configure the data to be monitored to send notifications, as well as the data whose value triggers a sending notification and the data sent into a notification.
- It is possible to configure several different notification types in parallel for the same or different applications FEs.
- It is possible to configure some feature behavior parameters, such as the destination of each notification message to be sent.

The parameters above are configured through the CUDB configuration Command Line Interface (CLI).

The notifications module runs in redundancy active-standby model. The active entity monitors the database events to be notified by subscribing to the master replicas of the DSGs hosted by the CUDB node. When the active entity fails, the standby entity becomes active and monitors the events instead of the active. For more information on high availability, refer to *CUDB High Availability*, Reference [3].

The module gathers internal statistics on failed and successful operations. These statistics are downloaded to counters, which are managed by the Ericsson Simple Network Management Protocol (SNMP) Agent (ESA). Additionally, the module produces logging information about the performed operations and the operation errors.

2.3 Description

The notifications feature has the following three types of attributes:

- **Monitored:** a notification is sent when the value of at least one attribute changes, considering that all of the monitored attributes must refer to the same LDAP entry or set of LDAP entries defined with a regular expression pattern for any defined notification.

The following events are considered as changes:

- Changing a value to other than the previous one.



- In non-multi-valued attributes, changing from no-value (`null` in the database) to a specific value.
- In non-multi-valued attributes, changing from a specific value to no-value (`null` in the database), if the object class containing the attribute is not removed. If the object class is removed, this event is not considered as a change, as this implies that the attribute is also removed.
- In multi-valued attributes, when at least one value is added to or deleted from the attribute, or when any of the values from the attribute are replaced with a new value.

It is possible to monitor the creation or deletion of not only the attributes but the entries too, by using the specific `monitorAll` option. See Section 3.2.2 on page 10 for further information.

- **Checked:** A notification is sent if at least one of the attributes has a certain and configured value, which must be different from no-value. Multi-valued attributes are not allowed to be checked.
- **Related:** The attributes to be included in the notification message. The notification is sent if the attributes are defined in the database and have a value (that is, if they are not null). When a multi-valued attribute is included in a notification, all the values currently held in the attribute are sent in it.

It is possible to define as many notifications as needed, but each one of them must have at least one monitored attribute. A notification is triggered if a monitored attribute changes and if the checked attributes (if defined) fulfill the comparison condition. All the attributes included in the notification are related to the same DE, as the feature is DE-based. Section 3.2.4 on page 18 shows an example of a notification definition.

A list of notification endpoints is also defined per notification. Each endpoint is characterized by a certain weight, which determines how often it receives notifications. Endpoints having a weight different from zero are included in a Weighted Round Robin (WRR) algorithm. This means, for example, if three endpoints (EP1, EP2 and EP3) are defined, each having weights 4, 2 and 1 respectively, then EP1 receives twice the number of notifications than EP2, and four times the number of notifications than EP3. On the other hand, if an endpoint has `weight=0` set, it is excluded from the WRR algorithm and receives all of the notifications. Therefore, all the endpoints with `weight=0` receive notifications in a broadcast. The order of WRR endpoints where notifications are being sent to is equal with the configuration order of the WRR endpoints in the configuration model.

2.4 Dependencies and Interactions

The notifications module works in an autonomous way in the CUDB system and does not interfere with the LDAP traffic operations performed by the system up



to the characterized performance limits. However, the following limitations in the feature and interactions with other features must be considered:

- In general, only binary content up to 4 KB is supported. Only string and integer attributes are supported in notifications (for more information about mapping LDAP to internal database types, refer to the related table in *CUDB Application Integration Guide*, Reference [4]). `OctetString` attributes are supported with the following limitations:
 - For monitor and related attributes, `OctetString` binary attributes up to 4 KB are supported.
 - For check attributes, `OctetString` binary attributes up to 1530 bytes are supported.
 - `OctetString` attributes greater than 4 KB are specifically not supported in any case.
- Depending on the feature configuration, unexpected notifications can be received during CUDB maintenance tasks, such as backup and restore operations and reallocation and reconciliation procedures because of deletion and creation of DEs in the database.
- PL data is not supported. The feature is limited to work with data stored in the DS only.
- In monitored attributes, changing from no-value (*null* in the database) to a specific value is considered a change.
- In monitored attributes, changing from a specific value to no-value is considered a change, except when the object class containing the attribute is also removed.
- Multi-valued attributes are not allowed to be included as checked values in the notifications.
- CUDB sends one notification message per LDAP operation that cause a change of any of the monitored attributes. However, consecutive LDAP modify operations received by CUDB within a very short period of time (about 1 second) over the same LDAP entry can be managed as only one LDAP operation according to the notifications feature. This sends just one notification message for all the modifications instead one per LDAP operation. In this case, the data in such notification message can contain inaccurate information in the sent multi-valued attributes.
- Operational attributes (`entryDS`, `structuralObjectClass`, `createTimestamp`, `modifyTimestamp`) are not supported in notifications.
- It is possible that the notification-related software processes show a high memory consumption. However, this situation is very unlikely, as it requires certain conditions to happen simultaneously, such as many notification



events configured, high notifications rate and several notification endpoints showing a slow response time.

- The maximum number of available TCP connections towards the notifications endpoints is 1000. This must be taken into account as an upper limit when configuring the Notifications feature. The number of needed TCP connections is calculated as follows:

$$\text{numOfConnections} = 20 * (\text{numNotifEventsWith1EP} * 1 + \text{numNotifEventsWith2EP} * 2 + \text{numNotifEventsWith3EP} * 3 + \dots)$$

In the above formula, **numOfConnections** is the number of needed TCP connections, while **numNotifEventsWith<X>EP** is the number of configured notifications events having <x> endpoints. See the below examples for more information.

– *Example 1*

In case of 3 notifications events with 2 endpoints defined, and 5 notifications events with 4 endpoints defined, the number of needed TCP connections is calculated as follows:

$$\text{numOfConnections} = 20 * (3 * 2 + 5 * 4) = 20 * 26 = 520$$

– *Example 2*

In case of 2 endpoints configured per event, 25 configured events are supported.

- When Data Repair is invoked, it may perform modifications to LDAP entries. In such case, notifications will be sent just like as if the modification had been performed by a node external to CUDB, such as by PG or any application FE.

Furthermore, there are some limitations about the lost notifications when the active instance dies. This loss is due to:

- AMF Switchover time.
- Time to initialize the process and the NDB subscription.
- Internal queues of the process.





3 Operation and Maintenance

This section describes the Operation and Maintenance of the notifications feature.

For information on troubleshooting this feature, refer to *CUDB Troubleshooting Guide*, Reference [5].

3.1 Activation

This section describes the activation of the notifications feature.

3.1.1 Prerequisites

The notifications feature is activated through the CUDB configuration model. The necessary prerequisites and permissions are listed in *CUDB Node Configuration Data Model Description*, Reference [6].

3.1.2 Activating Notifications

The notifications feature activation status is determined by attribute `enabled` in class `CudbNotifications` of the CUDB configuration model. If this boolean attribute is set to *false*, the sending notification is avoided.

3.2 Configuration

The notifications module is configured by a set of classes and attributes defined in the CUDB configuration model to provide the feature configuration options.

3.2.1 Configuring Notifications

The types of configuration parameters related to the notifications module are as follows:

- **Module configuration parameters:** applicable for the whole module. They can be modified at runtime, taking immediate effect after they are validated, without the need of disabling the feature and then enabling it again. Module configuration parameters are as follows:
 - Attribute `enabled`, class `CudbNotifications`: Enables or disables the notifications feature.



- Attribute `maxReattempts`, class `CudbNotifications`: Sets the number of reattempts to resend a notification in case of error.
- Attribute `reattemptTime`, class `CudbNotifications`: Sets the time between the attempts.
- **Notification related parameters:** applicable for each defined notification, not to the whole feature. Notification related parameters are as follows:
 - Endpoint definition parameters, detailed in class `CudbNotificationEndPoint`. Endpoint address and weight are defined here. These parameters can be modified at runtime, taking immediate effect after they are validated, without the need of disabling the feature and then enabling it again.
 - Monitored, checked and related attributes of the notification, defined in classes `CudbNotificationObjectClass` and `CudbNotificationAttr`. These parameters can be modified at runtime, but they need disabling and then enabling the feature again in order to take effect, which results in that notifications are not sent during this period.

See Section 3.2.2 on page 10 for further details on the parameters and classes, as well as the relationship and logic among them.

- **Security related parameters:** It is possible to configure secure endpoints by specifying TLS key and certificate as configuration parameters for the notifications. For more information, refer to *CUDB Security and Privacy Management*, Reference [2].

3.2.2 Notifications Parameters

`CudbNotifications` is the root class hosting all the configuration data related to the notifications feature. `CudbNotifications` contains the module configuration parameters, and all notifications defined in the CUDB system hang on it.

Notifications are defined by the `CudbNotificationEvent` class. This class has the same number of instances as the number of notifications created in the system and each instance contains the configuration data related to each notification, which is divided into the following two parts:

- Endpoint definition parameters are detailed in the `CudbNotificationEndPoint` class. The class has the same number of instances as the number of the notification endpoints, and each instance contains the related data for each endpoint.
- Notification object classes are detailed in the `CudbNotificationObjectClass` class. One instance of this class has to be defined per one LDAP entry, or per set of the LDAP entries that can be defined with one Portable Operating System Interface (POSIX) extended regular expression, hosting



the object classes that contain LDAP attributes involved in the notification. Instances of these classes can be the following types:

- `monitor`: LDAP attributes in `CudbNotificationObjectClass` instances of the `monitor` type are monitored in the notification. Monitored attributes can be configured for a specific entry or for a set of entries.

The `name` attribute contains the object class of these LDAP attributes.

The `dn` attribute must contain the Distinguished Name (DN), with or without the DE DN part, of the entry for which notification will be triggered. It can be configured in two different ways, as follows:

- For a specific entry, by setting the specific names and values of the DN.
- For a set of entries, by replacing parts of the DN or even the whole DN with POSIX extended regular expressions matching the entries for which to receive notifications.

When the whole DN (including the DE DN part) is used to configure the `dn` attribute, POSIX extended regular expressions can be also used to replace exact subscribers IDs.

All `dn` fields of all `CudbNotificationObjectClass` instances of the `monitor` type must contain the same value under the same instance of `CudbNotificationEvent`, so that all the monitored attributes refer to the same LDAP entry or set of LDAP entries.

See Section 3.2.3.2 on page 16 for specific examples about how to configure `CudbNotificationObjectClass` instances of the `monitor` type to monitor specific entries or a set of them.

One or several instances of class `CudbNotificationsAttr` can be created under each `CudbNotificationObjectClass` instance of the `monitor` type. All of them contain the attributes to be monitored within the object class. If any of them changes, the notification is subject to be sent, except when the attribute is single-valued and its object class is added or deleted in the same LDAP operation that changes the attribute value.

- `monitorAll`: This value has the same meaning and use as the `monitor` type, but it also triggers the sending notification when the attribute is single-valued and its object class is added or deleted in the same LDAP operation that changes the attribute value. Therefore, LDAP attributes contained in `CudbNotificationObjectClass` instances of the `monitorAll` type are monitored in the notification without exceptions. It monitors the creation or deletion of entries, not only the attributes. For more information, refer to *CUDB Node Configuration Data Model Description*, Reference [6].



- **check:** LDAP attributes in `CudbNotificationObjectClass` instances of the `check` type are checked to have the specified value to send the notification. The `dn` attribute must contain the DN of the entry without the DE DN part and the object class of these LDAP attributes.

One or several instances of class `CudbNotificationsAttr` can be created under each `CudbNotificationObjectClass` instance of the `check` type. All of them contain the attributes to be checked, and the value for the comparison.

If any of the check conditions is fulfilled the notification is sent. A condition is not fulfilled if the attribute to be checked has no-value.

- **related:** LDAP attributes contained in `CudbNotificationObjectClass` instances of the `related` type are included in the notification. The `dn` attribute must contain the DN of the entry without the DE DN part and the object class of these LDAP attributes.

One or several instances of class `CudbNotificationsAttr` can be created under each `CudbNotificationObjectClass` instance of the `related` type. Each instance contains the attributes to be sent in the notification.

The attributes in the *CUDB Node Configuration Data Model Description*, Reference [6] must be encoded as follows:

- Character strings for non-binary attributes.
- Base64-encoded strings for binary attributes.

At least one instance of the `CudbNotificationObjectClass`, `monitor`, or `monitorAll` types must be defined per notification. But it is possible not to define the `check` and `related` instance types. The CUDB system does not check at configuration time if the configured attributes belong to one of the supported types (see Section 2.4 on page 5).

The `send` field of the defined instances of `CudbNotificationAttr` acts as a sending flag for the related attribute. If it is true, the attribute name and value are sent in the notification. For multi-valued attributes, the complete set of values are sent. In case the attributes are included in the `CudbNotificationObjectClass` instances of the `monitor` or `monitorAll` types, the old attribute values are also sent.

If any of the LDAP attributes and object classes in the notification (either `monitor`, `monitorAll`, `checked` or `included` attributes) can not be retrieved from the database, the notification is not sent, and an error is reported.

For more information about these parameters, and generic SAF configuration model parameter handling directives, refer to *CUDB Node Configuration Data Model Description*, Reference [6].



3.2.3 Configuration Examples

This section provides different examples of configuring new CUDB Notification Events using regular expressions.

3.2.3.1 Creating a New Notification Event

To create a new CUDB Notification Event, a new instance of the `CudbNotificationEvent` class must be added to the configuration model and the corresponding attributes must be set. For more information, refer to the “Class `CudbNotificationEvent`” section of *CUDB Node Configuration Data Model Description*, Reference [6].

Example 1, shown in COM CLI, presents `CudbNotificationEvent=1` with `eventId="SAE-HLR"` with defined one Notification Endpoint and five Notification Object Classes:

```
>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1,CudbNotificationEvent=1
CudbNotificationEvent=1
  cudbNotificationEventId="1"
  eventId="SAE-HLR"
  notificationString="mobilityEvent"
  userLabel=[] <empty>
  CudbNotificationEndPoint=1
  CudbNotificationObjectClass=1
  CudbNotificationObjectClass=2
  CudbNotificationObjectClass=3
  CudbNotificationObjectClass=4
  CudbNotificationObjectClass=5
```

Example 1 *CudbNotificationEvent=1 with eventId="SAE-HLR"*

Define all required elements (attributes, object/object classes and their attributes) of the new `CudbNotificationEvent` in its tree structure (Directory Information Tree, DIT) before starting to create a new Notification Event.

Example 2 presents a new `CudbNotificationEvent=2` shown in COM CLI.



```

>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1,CudbNotificationEvent=2
CudbNotificationEvent=2
  cudbNotificationEventId="2"
  eventId="SAE-HSS"
  notificationString="mobilityEvent"
  userLabel= [] <empty>
  CudbNotificationEndPoint=1
  CudbNotificationObjectClass=1

>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1,CudbNotificationEvent=2,CudbNotificationEndPoint=1
CudbNotificationEndPoint=1
  name="Serv1"
  URI="http://127.0.0.1:8080"
  webService="/"
  weight=1

>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1,CudbNotificationEvent=2,CudbNotificationObjectClass=1
CudbNotificationObjectClass=1
  cudbNotificationObjectClassId="1"
  dn="serv=csp"
  name="CsPsLocationData"
  type="related"
  userLabel= [] <empty>
  CudbNotificationAttr=1

>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1,CudbNotificationEvent=2,⇒
CudbNotificationObjectClass=1,CudbNotificationAttr=1
CudbNotificationAttr=1
  cudbNotificationAttrId="1"
  name="SGSNUM"
  send=true
  userLabel= [] <empty>
  value=""
(CudbNotificationObjectClass=1)>

```

Example 2 New CudbNotificationEvent=2 shown in COM CLI

As shown in Example 2, CudbNotificationEvent=2 will contain one CudbNotificationEndPoint=1 and one CudbNotificationObjectClass=1 with one CudbNotificationAttr=1.

Follow the steps below, shown in COM CLI, to insert a new Notification Event using the Object Model Modification procedure:

1. Establish a CUDB CLI session towards the CUDB node by executing the `ssh -l cudbadmin <CUDB_Node_OAM_IP_Address>` command.

Important: This procedure must be done in all CUDB nodes in the system.

2. If there is no backup of the present configuration, perform the backup by executing the `sudo cmw-configuration-persist` command. If the backup is already updated, proceed to Step 3.

3. Establish a CUDB configuration CLI session in the active SC. Use the commands below (their output is also shown) to find the active SC:

```

sudo cudbHaState | grep COM | grep ACTIVE
COM is assigned as ACTIVE in controller SC-1

```

The active SC, (SC_2_1 in the example above) must be used for accessing the COM CLI by executing the `sudo /opt/com/bin/cliss` command.

4. Set the configuration session by executing the `configure` command.



5. Check if CudbNotifications are already enabled and exist in CudbNotificationsEvents.

Use the `show verbose` command to show `ManagedElement=1, CudbSystem=1, CudbNotifications=1`:

```
(config)>show verbose ManagedElement=1,CudbSystem=1,CudbNotifications=1
CudbNotifications=1
cudbNotificationsId="1"
enabled=true <default>
maxReattempts=3 <default>
reattemptTime=1000 <default>
userLabel=[] <empty>
CudbNotificationEvent=1
(config)>
```

If CudbNotifications are not already enabled, execute the following commands:

```
(config)>ManagedElement=1,CudbSystem=1,CudbNotifications=1,enabled=true
(config)>commit
```

6. Execute the following command sequence to add a new CUDB Notification Event (CudbNotificationEvent=2) in the CUDB configuration model:

Note: This is an example and must not be taken as a rule.

```
>configure
(config)>ManagedElement=1,CudbSystem=1,CudbNotification
s=1,CudbNotificationEvent=2
(config-CudbNotificationEvent=2)>eventId=SAE-HSS
(config-CudbNotificationEvent=2)>notificationString=mo
bilityEvent
(config-CudbNotificationEvent=2)>CudbNotificationEndP
oint=1
(config-CudbNotificationEndPoint=1)>name=Serv1
(config-CudbNotificationEndPoint=1)>URI=http://127.0.
0.1:8080
(config-CudbNotificationEndPoint=1)>weight=1
(config-CudbNotificationEndPoint=1)>up
(config-CudbNotificationEvent=2)>CudbNotificationObjec
tClass=1
(config-CudbNotificationObjectClass=1)>dn="serv=csps"
(config-CudbNotificationObjectClass=1)>name=CsPsLoca
tionData
(config-CudbNotificationObjectClass=1)>type=related
(config-CudbNotificationObjectClass=1)>CudbNotificati
onAttr=1
(config-CudbNotificationAttr=1)>name=SGSNNUM
(config-CudbNotificationAttr=1)>send=true
(config-CudbNotificationAttr=1)>up
```



```
(config-CudbNotificationObjectClass=1)>up
(config-CudbNotificationEvent=2)>show verbose
CudbNotificationEvent=2
cudbNotificationEventId="2"
eventId="SAE-HSS"
notificationString="mobilityEvent"
userLabel=[] <empty>
CudbNotificationEndPoint=1
CudbNotificationObjectClass=1
```

Each operation is executed as a unique transaction.

7. Commit the changes by executing the `commit` command.
8. Check the log files to see the result of the operations. For more information, refer to *CUDB Node Logging Events*, Reference [7].
9. Check the configuration changes with the `show ManagedElement=1,CudbSystem=1,CudbNotificationEvent=2` command.

Note: Remember to use `show verbose` instead of `show` for not mandatory attributes that must have no set value, or for optional attributes whose value is set to the default one.

10. Exit from the configuration mode by executing the `end` command.
11. Exit from the CUDB configuration CLI console by executing the `exit` command.

Note: Configuration changes in the examples above can also be performed through the NETCONF client.

See Section 3.2.2 on page 10 for more information about Notifications Parameters.

Refer to the Object Model Modification Procedure in *CUDB Node Configuration Data Model Description*, Reference [6] for more information on all the steps required to modify the object model (for example, on using the administrative operation `applyConfig` to activate the changes).

3.2.3.2 Using Regular Expressions

For using regular expressions to define `monitor type CudbNotification ObjectClass` classes, see Example 3, Example 4, Example 5, Example 6, and Example 7 .



Warning!

When using regular expressions to configure a partial DN, the configured regular expression is always compared to whole DN of modified entry, so using the “.” wildcard at the end of regular expression is mandatory.

```
CudbNotificationObjectClass=1
cudbNotificationObjectClassId="1"
dn="EpsDynInfId=EpsDynInf,EpsStaInfId=EpsStaInf,serv=eps,mscId=.*,ou=multiSCs,dc=operator,dc=com"
name="EpsDynInf"
type="monitor"
userLabel=[] <empty>
CudbNotificationAttr=1
```

Example 3 Using Regular Expression to Define Whole DN of LDAP Entry

Notification will be triggered for any change of defined monitored attribute(s) in the EpsDynInf object class for every mscId.

```
CudbNotificationObjectClass=2
cudbNotificationObjectClassId="2"
dn="(grpId=.*,serv=pcrf).*"
name="PCRFServiceGroup"
type="monitor"
userLabel=[] <empty>
CudbNotificationAttr=1
```

Example 4 Using Regular Expressions in dn Attribute with Partial DN (without DE DN Part)

If the configuration in Example 4 must be restricted to only few values of grpId, the OR operator can be used.

```
CudbNotificationObjectClass=2
cudbNotificationObjectClassId="2"
dn="( (grpId=id1|grpId=id2),serv=pcrf).*"
name="PCRFServiceGroup"
type="monitor"
userLabel=[] <empty>
CudbNotificationAttr=1
```

Example 5 Using Regular Expressions with logical operators, in dn Attribute with Partial DN (without DE DN Part)

Or, to accomplish the same:

```
CudbNotificationObjectClass=2
cudbNotificationObjectClassId="2"
dn="(grpId=(id1|id2),serv=pcrf).*"
name="PCRFServiceGroup"
type="monitor"
userLabel=[] <empty>
CudbNotificationAttr=1
```

Example 6 Using Regular Expressions with logical operators, in dn Attribute with Partial DN (without DE DN Part)

Notification will be triggered for any change of defined monitored attribute(s) in the PCRFServiceGroup object class for any value of grpId.



```
CudbNotificationObjectClass=3
cudbNotificationObjectClassId="3"
dn="recordId=.*,viewId=.*,fqdn=.*,ou=EnumDn.*,serv=enum,ou=servCommonData,dc=operator,dc=com"
name="NAPTRRecord"
type="monitor"
userLabel=[] <empty>
CudbNotificationAttr=1
```

Example 7 Using Multiple Wildcards in dn Attribute with Whole DN

Notification will be triggered for any change of defined monitored attribute(s) in the NAPTRRecord object class for any entry whose DN matches the defined pattern.

Note: Any regular expression can be used to define the DN pattern. However, complex regular expressions should be used with caution to avoid unexpected notifications.

3.2.4 Preconfigured Notifications

An Evolved Packet Core (EPC) Mobility Notification is preconfigured in CUDB. It sends notifications to the Home Location Register (HLR) FE. EPC mobility support is an optional CUDB feature, as detailed in *CUDB Technical Product Description*, Reference [8].

The following attributes are involved in the preconfigured notification:

- Monitored: EpsMobilityNotifInfo
- Checked: PSLOC, which must be equal to *SGSN Known* or *MSC Purged in SGSN*.
- Related:
 - International Mobile Subscriber Identifier (IMSI).
 - Serving GPRS Support Node (SGSN) node where the subscriber is located.
 - Collision Detection Controller (CDC).

3.2.5 External Networks Connectivity

As notifications are sent outside the CUDB, it is needed to configure the network connectivity facilities in the system for communication towards the endpoints. For more information, refer to *CUDB Node Network Description*, Reference [9].

Additionally, eVIP routes for the endpoints must be added in the blades or Virtual Machines (VMs) where the notification processes are running, if they are not already established. Otherwise, the endpoints cannot be accessed. For more information, refer to *CUDB System Administrator Guide*, Reference [10].



It is also possible to configure how TCP sockets towards the endpoints are initialized per notification. For more information about configuring socket initialization, refer to *CUDB SOAP Interwork Description*, Reference [1].

3.3 Fault Management

This feature does not trigger any alarms.

3.4 Performance Management

For more information on counters managed by the notifications module, refer to *CUDB Counters List*, Reference [11].

3.5 Security

It is possible to configure a TLS key and certification in the SOAP interface. For detailed information about security related parameters and security configuration, refer to *CUDB Security and Privacy Management*, Reference [2].

3.6 Logging

For more information on the log messages generated by the notifications module, refer to *CUDB Node Logging Events*, Reference [7].





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 SOAP Interwork Description*
- [2] *CUDB Security and Privacy Management*
- [3] *CUDB High Availability*
- [4] *CUDB Application Integration Guide*
- [5] *CUDB Troubleshooting Guide*
- [6] *CUDB Node Configuration Data Model Description*
- [7] *CUDB Node Logging Events*
- [8] *CUDB Technical Product Description*
- [9] *CUDB Node Network Description*
- [10] *CUDB System Administrator Guide*
- [11] *CUDB Counters List*
- [12] *CUDB Glossary of Terms and Acronyms*