

CUDB Application Counters

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	Scope	1
1.2	Revision Information	1
1.3	Target Groups	2
1.4	Prerequisites	2
1.5	Typographic Conventions	2
1.6	Related Terms	2
2	Overview	3
2.1	Architecture	3
2.2	Description	4
2.3	Dependencies and Interactions	5
3	Operation and Maintenance	7
3.1	Configuration	7
3.2	Fault Management	12
3.3	Performance Management	12
3.4	Security	12
3.5	Logging	12
4	Appendix: Examples	13
4.1	Creating Application Counters Configuration File	13
4.2	Creating Tables	13
4.3	Stored Procedures	13
4.4	ESA Configuration	15
4.5	Configuring Cron File: appCounters.cron	17
	Glossary	19
	Reference List	21





1 Introduction

This document provides a description of the application counters function in the Ericsson Centralized User Database (CUDb) and describes how to create and configure application counters by using the Application Counters Framework.

1.1 Scope

The purpose of this document is to describe the application counters function. The application counters function is required by applications storing their data in CUDb to show statistics generated from the stored data.

1.2 Revision Information

This section contains the changes in the feature between the releases of this document.

Rev. A

This document is based on 10/155 34-HDA 104 03/9 with revised content and structure.

Rev. B

Editorial changes only.

Rev. C

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

- Section 3.1.4.3 on page 10: Removed note.
- Section 3.1.5 on page 11: Updated the procedure.

Rev. D

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

- Updated Ericsson personnel information.

Rev. E

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

- Section 3.1.4.3 on page 10: Updated the configuration procedure.



1.3 Target Groups

This document is intended for system administrators.

1.4 Prerequisites

Users of this document must have knowledge and experience of the following:

- CUDB internal way of working, processes and file structure.
- Ericsson SNMP Agent (ESA) configuration and behavior.
- Logic of the application and its data model to create files needed to generate its counters.

1.5 Typographic Conventions

Typographic conventions can be found in the following document:

- [Typographic Conventions](#)

1.6 Related Terms

This section is not applicable to this feature.



2 Overview

The application counters function allows applications to create counters related to the application data stored in CUDB. Counter values are stored in CUDB tables, which are periodically consolidated and written to 3GPP standards-compliant EXtended Markup Language (XML) files, the officially supported format.

2.1 Architecture

Application counters are generated at system level, but are published by all CUDB nodes. This means that while each application counter shows the situation considering the whole CUDB system, they can be obtained from any of the CUDB nodes.

Note: Application Counters values obtained from nodes with slave PLDB may differ from the values in the node with the master PLDB replica as a consequence of the asynchronous replication.

Figure 1 shows interactions between application counters process and the rest of the components in the CUDB system.

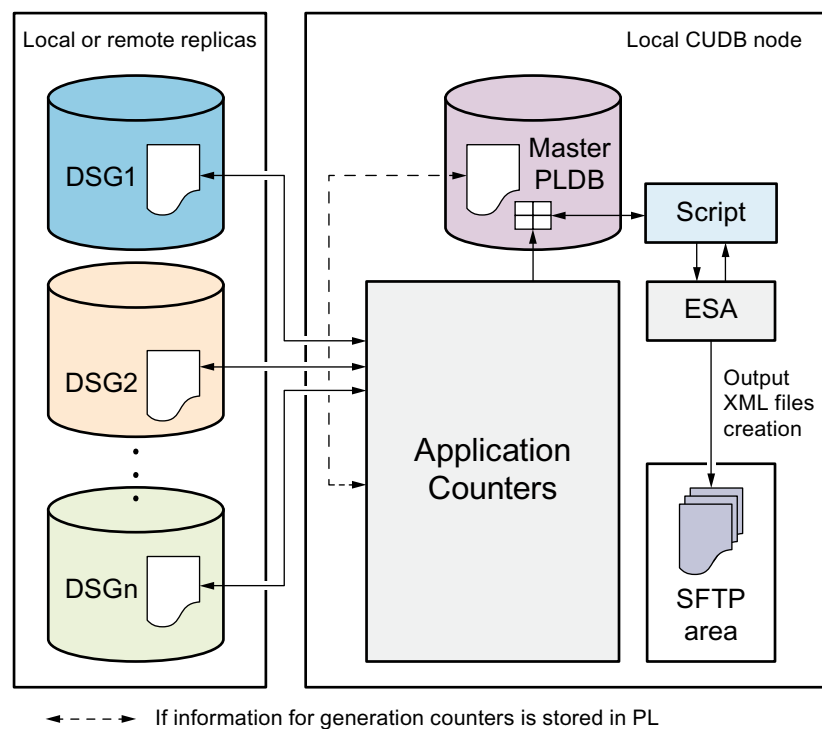


Figure 1 Application Counters Overview



Application counters are generated by collecting information from replicas of all Data Store Unit Groups (DSGs) in the whole CUDB system (or from PLDBs in case of counters related to information stored in a PLDB), and published in XML files. The XML files can then be fetched through Secure File Transfer Protocol (SFTP).

2.2 Description

The Application Counters function supports two major tasks, counter generation and publishing.

— Counter generation performs the following:

- 1 A task per counter group configured in Linux `cron` scheduler is started.

Note: `cron` tasks that start the application counters process must be defined after all configuration files are ready.

- 2 This task connects to PLDB or to one DS Unit per DSG, and executes an SQL procedure.

Note: If the DSG has more than one DSG replica and the stored procedures cannot be executed on the selected replica (for example, after a restore has been performed and the application counters were not installed again), then the application counters process tries to execute the procedures on other available replicas.

- 3 Application counters process adds up the values received from PLDB or the values of each counter received from all DSGs and it stores them in a table in PLDB.

Note: The replicas generating the counters can either be DSG slaves or the PLDB master replica, that is, the PLDB replica of the CUDB node where the Application Counters process is running. The replica selection depends on whether the values need to be calculated from the DS or from the PLDB.

— Publishing performs the following:

- 1 Periodically, ESA uses the shell scripts indicated in its configuration file to fetch the counters from PLDB.
- 2 Periodically, ESA writes the values of the counters in XML files, according to the jobs defined.

Note: The nodes that calculate the values can either be DSG slaves or the PLDB master replica, depending on whether the values are calculated from DSGs or PLDB.

If an error occurs while fetching counter values, CUDB raises the `Application Counters, Fault In Subscriber Statistic Application, Reference [1]` alarm.



2.3 Dependencies and Interactions

Stored procedures are lost and must be re-installed in that cluster in case of the following circumstances:

- System data restore is executed.
- Group or unit data restore is executed using `cudbManageStore` command without the option `--restore-stored-procedures`.
- Cluster is restarted with `cudbManageStore --order initialize`.

Because application counter values are fetched from the Application Counters DB and exposed in the Lightweight Directory Access Protocol (LDAP) Directory Information Tree (DIT) under a separate root entry, these values are not available in the DIT until the stored procedures are re-installed.





3 Operation and Maintenance

This section describes the operation and maintenance of the Application Counters function.

3.1 Configuration

To configure CUDB application counters, the following steps are followed:

- Create application group configuration file for each counter group.
- Create the tables to store the counters in PLDB.
- Create the stored procedures to calculate the counters. A `cron` task executes them to fetch the values of the counters.
- Configure ESA.
- Configure the `cron` task that gathers the counters so it knows how to fetch them and where to store them.

3.1.1 Application Counters Configuration File

The application counters configuration file contains the information needed for generating counters and storing them in PLDB.

The following information is included in this configuration file:

- An optional label to indicate if the process must only count values from PLDB or it must add up values from each DS group.
- Name of the stored procedure to call.

Note: This is also the name of the PLDB table where counters must be stored.

- Name of the group of counters and the name of the table.
- Name of all counters (one in each line) included in the stored procedure. It is also possible to specify input parameters for the stored procedures in each counter: `<paramValue>`

Note: This is also the name of the columns of the PLDB table where each counter is stored.

The format of the configuration file, per counter group, is as follows:

```
[*PL_COUNTER*]
<procedure_name>
```



```
<table_name>
<counter_1_name>=<paramValue1>#<paramValue2>#...#<paramValueN>
<counter_2_name>=<paramValue>
...
<counter_n_name>
```

The configuration file must be stored in `/home/cudb/oam/performanceMgmt/appCounters/config/` with extension `.conf`.

Example 1 shows a sample application counters configuration file.

3.1.2 Create Tables in PLDB

Tables for the application counters must be created in the `cudb_application_counters` database, in PLDB. The SQL scripts that create these tables are stored under `/home/cudb/oam/performanceMgmt/appCounters/schema/` and the files have `.sql` extension.

It is recommended to use each table for a group (collection) of counters.

Note: Defining a primary key for the Application Counters tables is mandatory. For example, it can be the first column of the table.

Note: Using the same counter and group names that are used in ESA is recommended as it is simpler to maintain all the configuration files for a counter (or a group of counters). For further information about the ESA collection of counters, refer to [ESA Performance Management](#), Reference [8].

The tables must be filled with initial values and created in a single CUDB node on which PLDB runs as Master.

Before fetching the counter values, wait until these tables are replicated in all PLDBs in the CUDB system. See Example 2.

Once the `.sql` file is created, execute it on an SC as follows :

```
shell> mysql -h<pl0,pl1> -P15000 --user=<user_name> --password=<password> -e "source <complete_path_to_applicationCountersTable.sql>"
```

In the above command, `<user_name>` and `<password>` are obtained from CUDB Users and Passwords, Reference [2].

Note: All tables must be created only from a single PLDB (PL0 or PL1), as these tables depend on the database cluster and are visible from other PLDBs.



3.1.3 Create Stored Procedures

The scripts with the stored SQL procedures must be kept in the following location on all CUDB nodes:

```
/home/cudb/oam/performanceMgmt/appCounters/procedures/
```

The database where the procedures are executed has to be specified at the beginning of the script, as shown in Example 3.

Note: Application counters are defined on relational tables and rows, not on LDAP trees, so if the application uses LDAP Data Views, it adds an extra level of indirection, which must be considered during the definition of the counter (Views LDAP tree -> core DIT tree -> tables and rows). For detailed information on LDAP Views, refer to [CUDB LDAP Data Views Management, Reference \[3\]](#).

Once the stored procedure file is created, execute it on all access servers of the PLDB or DSG clusters in the CUDB system (depending on whether the counter is contained in PLDB or DSG), necessary so that all needed stored procedures receive the complete set of application counters.

The command to execute on the database cluster access servers of PLDB is the following:

```
shell> mysql -h<pl0,pl1> -P15000 --user=<user_name> --password=
<password> -e "source <complete_path_to_applicationCountersProce
dure.sql>"
```

In the above command, `<user_name>` and `<password>` are obtained from CUDB Users and Passwords, Reference [2].

Note: The stored procedures must be created from both PL0 and PL1, as stored procedures are created in the database cluster access server and are not propagated to other PLDB.

The command to execute on the database cluster access servers of the DS is the following:

```
shell> mysql -h<dsX_0,dsX_1> -P<15000+10*X> --user=<user_name>
--password=<password> -e "source <complete_path_to_applicationCoun
tersProcedure.sql>"
```

In the above command, `<user_name>` and `<password>` are obtained from CUDB Users and Passwords, Reference [2].

Note: The stored procedures must be created from both DSX_0 and DSX_1, as stored procedures are created on the database cluster access server and are not propagated to other DS.

See Section 4.3.1 on page 14 to learn about the best practices in writing stored procedures.



3.1.4 Configure ESA

Accounting for counters, configure ESA with the following three steps:

- Create scripts to read the counters from the tables.
- Configure ESA to use those scripts to fetch the counters.
- Configure jobs to generate the output files.

Detailed instructions follow below.

3.1.4.1 Create Scripts to Fetch Application Counters to ESA

The shell scripts required for a group of counters are called by ESA. To ease the retrieval and formatting of counter data, the following shell library is provided with functions that scripts can use:

```
/home/cudb/oam/performanceMgmt/appCounters/scripts/appCountersLib.sh
```

In each script, a different query must be defined to retrieve different sets of application counters. Example 7 shows a sample script.

Changing the **QUERY** and **COUNTERID** variables is enough for these scripts to work. **QUERY** must be an SQL SELECT sentence retrieving all the counters in a group located in the same table.

The scripts must be stored in the following location for all CUDB nodes of the CUDB system:

```
/home/cudb/oam/performanceMgmt/appCounters/scripts/
```

3.1.4.2 Configure ESA to Use Scripts

An XML file must be created to configure ESA to fetch the application counters in `/home/cudb/oam/performanceMgmt/config/PmCounters` on all CUDB nodes. The description of the XML file can be found in [ESA Performance Management, Reference \[8\]](#). See Example 8.

Note: The default value for the **interval** attribute within tag **<dataSource>** is 900, representing the time interval, in seconds, between fetching the application counter values from PLDB.

3.1.4.3 Configure Jobs to Generate Output Files

Application counters output files are configured using XML-based job files placed in the following folder:

```
/home/cudb/oam/performanceMgmt/config/PmJobs
```



This folder is located on shared storage for all blades or VMs in CUDb. The job files within contain configuration parameters, such as the periodicity of generating counter files. Each counter group is configured with one job file.

The format of these files is described in *ESA Performance Management, Reference [8]*.

To configure jobs to generate output files, create the corresponding job files and place them in the above folder through SFTP established directly from the Network Management Systems (NMSs). For further information about output files location, naming, and format, refer to *CUDb Performance Guide, Reference [5]*.

3.1.5 Configure Cron Task

To generate application counters, tasks must be introduced in the `cron` of the SCs. This section describes the configuration file for this task and the steps required to configure `cron`.

The procedure to configure `cron` is as follows:

1. Modify the `/home/cudb/oam/performanceMgmt/appCounters/scripts/appCounters.cron` file to include the new task, following this format:
`/opt/ericsson/cudb/OAM/bin/cudbApplicationCounters -C
 /cluster/home/cudb/oam/performanceMgmt/appCounters/config/<app_counter>.conf -U 1 -u LOCAL1`

The `<app_counter>.conf` parameter corresponds to the configuration file created in Section 3.1.1 on page 7. For further information about the parameters of this executable, refer to *CUDb Node Commands and Parameters, Reference [4]*.

See Example 9.

Note: Using the `appCounters.cron` file as a template (see Section 4.5 on page 17), several `.cron` files can be defined, so specific execution times or periodicity can be defined independently for different counter groups. This also allows the activation or deactivation of certain groups of counters.

Note: The heading of the file must be `#!/bin/bash -l` to enable the use of environment variables.

2. To add `cron` execution time to the `.cron` file (for example `appCounters.cron`), edit the file by adding `cron` time expression to the `CUSTOM_SCHEDULE_TIME` variable. In case this variable has not been edited (left blank), the `cron` execution time will be set to the default time expression (`0,15,30,45 * * * *`).

Note: In case multiple `.cron` files are used, it is recommended to define the `CUSTOM_SCHEDULE_TIME` in a way that there is no parallel execution of application counters defined in different files. In typical application counters configuration, with a single `.cron` file, this is achieved by using the default time expression with the scheduling interval of 15 minutes.



3. Activate the new planned tasks in the operating system in both SCs on all CUDB nodes of the CUDB system with the following command:

```
# /etc/init.d/cudbappCounters start
```

Note: This command will import all the `.cron` files as `cron` tasks, with their `cron` time expressions as `cron` execution time, into `cron`. They can be reviewed using the `crontab -l` command on both SCs.

To stop the scheduled `cron` tasks in the operating system in both SCs on all CUDB nodes of the CUDB system with the following command:

```
# /etc/init.d/cudbappCounters stop
```

Note: This command will remove all the `.cron` files, as `cron` tasks, from `cron`. The result can be reviewed by using the `crontab -l` command on both SCs.

`.cron` file entries are added to `crontab` persistently at reboot, to remove files permanently from `crontab`, remove the files from their location.

3.2 Fault Management

During the execution of the application counters process, the following alarm can be raised:

— Application Counters, Fault In Subscriber Statistic Application

3.3 Performance Management

This section is not applicable to this feature.

3.4 Security

This section is not applicable to this feature.

3.5 Logging

During the processing of application counters, a number of events can be logged. For further information, refer to [CUDB Node Logging Events, Reference \[6\]](#).



4 Appendix: Examples

This section provides examples for the procedures described in this document.

4.1 Creating Application Counters Configuration File

Example 1 shows a sample application counters configuration file.

```
*PL_COUNTER*
GET_AUCSUBS_COUNTERS
GRP_AUCSUBS
NSUBSCNT
NGSUBSCNT
NUSUBSCNT
NUMSUBSPARAMS=30
```

Example 1 Application Counters Configuration File

4.2 Creating Tables

Example 2 shows creating a table in PLDB.

```
The stored procedures for each counter group will have
the same name as the group.
Therefore the procedure name for this group will be
GET_AUCSUBS_COUNTERS
*/
USE cudb_application_counters;
DROP TABLE IF EXISTS GRP_AUCSUBS;
CREATE TABLE GRP_AUCSUBS(
NSUBSCNT integer PRIMARY KEY,
NGSUBSCNT integer,NUSUBSCNT integer)
ENGINE=NDB;

INSERT INTO GRP_AUCSUBS VALUES (0,0,0);
```

Example 2 Creating a Table in PLDB

4.3 Stored Procedures

Example 3 shows stored procedures.



```

USE cudb_user_data;
DELIMITER //

DROP PROCEDURE IF EXISTS GET_AUCSUBS_COUNTERS //
CREATE PROCEDURE GET_AUCSUBS_COUNTERS (OUT NSUBSCNT INT,
OUT NGSUBSCNT INT, OUT NUSUBSCNT INT)

BEGIN
    SELECT COUNT(*) INTO NSUBSCNT FROM AuthIMSIData
    WHERE IMSI IS NOT NULL;

    SELECT COUNT(*) INTO NGSUBSCNT FROM AuthIMSIData
    WHERE AKATYPE = 0;

    SELECT COUNT(*) INTO NUSUBSCNT FROM AuthIMSIData
    WHERE AKATYPE = 1;

END //
DELIMITER ;

```

Example 3 Stored Procedures

4.3.1 Best Practices in Writing Stored Procedures

- Column order is not guaranteed in CUDB. Never use the column `id` in the **ORDER BY** clause when writing SQL queries. Instead of **SELECT ... ORDER BY 1,2**, write an expression or column name(s) in the **ORDER BY** clause.
- Use **EXPLAIN**, **EXPLAIN EXTENDED** and **EXPLAIN PARTITIONS** statements to see and understand the query execution plan, and to confirm that the expected index(es) are used for your query.
- Keep the number of full table scans to the minimum. Instead of 13 queries like the ones shown in Example 4, use one **SELECT** statement as shown in Example 5 on the same table to collect all count numbers.

```

SELECT COUNT(*) FROM CP6 WHERE AOC = 1;
SELECT COUNT(*) FROM CP6 WHERE AOC = 2;
SELECT COUNT(*) FROM CP6 WHERE HOLD IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE ICI IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE OIN IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE TIN IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE MPTY IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE CLIP IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE CLIR IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE COLP IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE COLR IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE OICK IS NOT NULL;
SELECT COUNT(*) FROM CP6 WHERE TICK IS NOT NULL;

```

Example 4 Optimizing Stored Procedures - 13 Queries



```

SELECT
SUM(IF(AOC=1, 1,0)) as value1,
SUM(IF(AOC=2, 1,0)) as value2,
SUM(IF(HOLD IS NOT NULL, 1,0)) as value3,
SUM(IF(ICI IS NOT NULL, 1,0)) as value4,
SUM(IF(OIN IS NOT NULL, 1,0)) as value5,
SUM(IF(TIN IS NOT NULL, 1,0)) as value6,
SUM(IF(MPTY IS NOT NULL, 1,0)) as value7,
SUM(IF(CLIP IS NOT NULL, 1,0)) as value8,
SUM(IF(CLIR IS NOT NULL, 1,0)) as value9,
SUM(IF(COLP IS NOT NULL, 1,0)) as value10,
SUM(IF(COLR IS NOT NULL, 1,0)) as value11,
SUM(IF(OICK IS NOT NULL, 1,0)) as value12,
SUM(IF(TICK IS NOT NULL, 1,0)) as value13
FROM CP6 WHERE AOC=1 OR AOC=2 OR HOLD IS NOT NULL OR ICI IS NOT NULL OR OIN IS NOT NULL OR TIN IS NOT NULL OR MP
NULL OR CLIP IS NOT NULL OR CLIR IS NOT NULL OR COLP IS NOT NULL OR COLR IS NOT NULL OR OICK IS NOT NULL OR TICK

```

Example 5 Optimizing Stored Procedures - Single SELECT Statement

- Keep the number of join-s to the minimum.
- Write queries that can be pushed down to the cluster.
- Optimize queries using the IN operator. See Example 6.

Instead of:

```

SELECT COUNT(*) INTO NACTSUBSCNT FROM CP2 WHERE CSLOC = 0 OR
CSLOC = 3 OR
CSLOC = 4 OR
(CSLOC = 5 AND ( DATE_SUB(NOW(), INTERVAL INTERVAL_DAY DAY) < STR_TO_DATE(hexStr2decStr(HEX(PURGE
'%y%m%d') ) ) OR
PSLOC = 0 OR
PSLOC = 3 OR
PSLOC = 4 OR
(PSLOC = 5 AND ( DATE_SUB(NOW(), INTERVAL INTERVAL_DAY DAY) < STR_TO_DATE(hexStr2decStr(HEX(PURGE
'%y%m%d') ) ) );

```

when possible use:

```

SELECT COUNT(*) INTO NACTSUBSCNT FROM CP2
WHERE CSLOC IN (0,3,4) OR
(CSLOC = 5 AND ( DATE_SUB(NOW(), INTERVAL INTERVAL_DAY DAY) < STR_TO_DATE(hexStr2decStr(HEX(PURGEDATECS)
) ) OR
PSLOC IN (0,3,4) OR
(PSLOC = 5 AND ( DATE_SUB(NOW(), INTERVAL INTERVAL_DAY DAY) < STR_TO_DATE(hexStr2decStr(HEX(PURGEDATEPS)
) ) );

```

Example 6 Optimizing Queries Using IN Operator

4.4 ESA Configuration

This section provide examples of ESA configuration.

4.4.1 Scripts to Fetch Counters

Example 7 shows a sample script to fetch counters.



```
#!/bin/bash

. /home/cudb/oam/performanceMgmt/appCounters/scripts/appCountersLib.sh

QUERY="select NSUBSCNT,NGSUBSCNT,NUSUBSCNT from GRP_AUCSUBS;"

#DEBUG=1
mysql_query "$QUERY"

# Simple solution is to exit on failure
# More critical counters could follow a retry policy
if [ $? != 0 ]
then
    logging_system 'error' "Query \"$QUERY\" failed with \"$ROWS\""
    exit 1
fi

GROUPID=AUCSUBS
COUNTERID=AUCSUBS
format_counter $GROUPID $COUNTERID

exit $?
```

Example 7 A Script to Fetch Counters

4.4.2 Configuring ESA to Use Scripts

Example 8 shows configuring ESA to use scripts.



```
<?xml version="1.0" encoding="UTF-8"?>
<pmCntGroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ericsson.com/esa"
xsi:schemaLocation="http://www.ericsson.com/esa pmCounter.xsd"
active="yes">
  <identification>
    <groupId>AUCSUBS</groupId>
  </identification>
  <description>
    <groupDescr>AUC application subscriber counters</groupDescr>
    <groupInfo>These counters permits evaluate the use of
certain functions of the AUC application</groupInfo>
  </description>
  <!--Counter collection with data source SCRIPT-->
  <!-- G1-->

  <cntCollection active="yes" activeSnmp="yes">
    <identification>
      <collectionId>AUCSUBS</collectionId>
    </identification>
    <cntDefinition cntType="Gauge">
      <identification>
        <counterId>NSUBSCNT
</counterId>
      </identification>
      <description>
        <counterDescr>Number of subscribers</counterDescr>
        <counterInfo/>
      </description>
    </cntDefinition>
    <cntDefinition cntType="Gauge">
      <identification>
        <counterId>NGSUBSCNT</counterId>
      </identification>
      <description>
        <counterDescr>Number of GSM subscribers</counterDescr>
        <counterInfo/>
      </description>
    </cntDefinition>
    <cntDefinition cntType="Gauge">
      <identification>
        <counterId>NUSUBSCNT</counterId>
      </identification>
      <description>
        <counterDescr>Number of UMTS subscribers</counterDescr>
        <counterInfo/>
      </description>
    </cntDefinition>
    <dataSource interval="900">
      <script>
<location>/cluster/home/cudb/oam/performanceMgmt/
appCounters/scripts/APCount_AUCSUBS.sh</location>
      </script>
    </dataSource>
  </cntCollection>
</pmCntGroup>
```

Example 8 Configuring ESA to Use Scripts

4.5 Configuring Cron File: appCounters.cron

Example 9 shows the configurable part of a sample appCounters.cron file.



```
#!/bin/bash -l
# ----- ONLY THIS PART OF THE SOURCE HAS TO BE MODIFIED -----
# -----

LIST_APP_COUNTERS[0]="/opt/ericsson/cudb/OAM/bin/\
cudbApplicationCounters \
-C /cluster/home/cudb/oam/performanceMgmt/appCounters/config/\
<app_counter1>.conf -U 1 -u LOCAL1"

LIST_APP_COUNTERS[1]="/opt/ericsson/cudb/OAM/bin/\
cudbApplicationCounters \
-C /cluster/home/cudb/oam/performanceMgmt/appCounters/config/\
<app_counter2>.conf -U 2 -u LOCAL1"

LIST_APP_COUNTERS[2]="/opt/ericsson/cudb/OAM/bin/\
cudbApplicationCounters \
-C /cluster/home/cudb/oam/performanceMgmt/appCounters/config/\
<app_counter3>.conf -U 3 -u LOCAL1"

LIST_APP_COUNTERS[3]="/opt/ericsson/cudb/OAM/bin/\
cudbApplicationCounters \
-C /cluster/home/cudb/oam/performanceMgmt/appCounters/config/\
<app_counter4>.conf -U 4 -u LOCAL1"
# -----
```

Example 9 appCounters.cron File



Glossary

For the terms, definitions, acronyms and abbreviations used in this document, refer to [CUDB Glossary of Terms and Acronyms](#), Reference [7].





Reference List

CUDB Documents

- [1] Application Counters, Fault In Subscriber Statistic Application
- [2] CUDB Users and Passwords, 3/00651-HDA 104 03/10
- [3] CUDB LDAP Data Views Management
- [4] CUDB Node Commands and Parameters
- [5] CUDB Performance Guide
- [6] CUDB Node Logging Events
- [7] CUDB Glossary of Terms and Acronyms

Other Ericsson Documents

- [8] ESA Performance Management