

Database Access

Ericsson Service-Aware Policy Controller

User Guide

Copyright

© Ericsson AB 2018. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

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

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document [Trademark Information](#).



Contents

1	Database Access Introduction	1
2	Database Access Function	2
2.1	Database Access Function Overview	2
2.2	Supported external databases Technologies	4
2.2.1	Authenticated Connections to the external database	4
2.3	Data Model Mapping	4
2.3.1	Entities Relations	6
2.3.2	Entity Instance Searches	8
2.3.3	Use of Data to Evaluate Policy Conditions	9
2.4	Write Operations	10
2.5	Data Combinations Use Cases	10
2.5.1	Subscriber Data	10
2.5.1.1	Subscriber Data Extension in External Database	11
2.5.2	Data Extension in the SAPC internal database	11
2.5.3	Distributing Same Data in Different Repositories	12
2.5.4	Mixing Different Types of Repositories for An Application object	13
2.5.5	Conditions on external database Data Model	14
2.6	Performance Optimizations	15
2.6.1	LDAP Repositories	15
2.7	Fallback Mechanism for Subscribers in External Databases	16
3	Database Access Operation and Maintenance	17
3.1	Configure Entity Data Sources	17
3.1.1	Configure Keys for Application object Searches	17
3.1.1.1	Use of Arguments in Policy Conditions	18
3.1.2	Data Restrictions	20
3.1.3	Configure to Write	22
3.1.4	Configure Fallback for Subscribers	22
3.2	Configure LDAP User Authentication	22
3.3	Configure Use of LDAP Scope, Filter, Alias Dereferencing	23
3.4	Database Access Configuration Examples	24
3.4.1	Using Functions to Modify the Subscriber Identifier	24
3.4.2	Configure Extra Data In internal database	24
3.4.3	LDAP Configuration Examples	26
3.4.3.1	New Subscriber Data in an External LDAP Repository	26
3.4.3.2	New Entity Stored in an External LDAP Repository	28
3.4.3.3	Subscriber Distributed across Several LDAP Repositories	29



3.4.4	Configure New Entity with Constants	32
3.4.5	Configure Subscriber Data Extension In the SAPC internal database	33
3.4.6	Configure Fair usage accumulators for Read and Write	33
3.4.7	Configure Session Context for Read and Write	34
3.4.8	Examples of Policy Conditions Using Entity Data Source	34
3.4.8.1	Example Using Data Expanding Subscriber	34
3.4.8.2	Example Using New Entity	34
3.4.8.3	Example Using a Constant	35
3.5	Entity Data Definition Language	35
3.6	Database Access Fault Management	40
3.6.1	Database Access Alarms	40
3.6.2	Database Access Notifications	41
3.7	Database Access Logging	41
3.8	Database Access Performance Management	41
4	Reference List	42



1 Database Access Introduction

This document contains the description of Database Access function, and also the details on how to administer it in the SAPC.

Some of the data of the SAPC can be physically stored either in the SAPC internal database or in an external database.

Owing to the significant influence of the SAPC configuration in External Database function, this document contains two complementary purposes:

- It describes the functionality that the SAPC offers regarding its possibilities to ease the integration with the operator external database. In this sense, this document is a functional description.
- Understand how to manage (operate) the Database Access. In this sense, it is a User Guide.

This document is not an exhaustive guide for configuring the SAPC in every possible scenario.



2 Database Access Function

2.1 Database Access Function Overview

There are two different uses of Database Access in the SAPC:

- Application objects: The SAPC uses different data to perform the tasks associated to its business logic. Therefore, the SAPC defines an application domain information model with all the data that is needed. This information model specifies how the different data elements are grouped into data structures and how these data structures relate to each other.

The Database Access function allows that the Subscriber-related data can be physically stored either in the SAPC internal database or in an external repository.

- Data that the SAPC uses in policy conditions evaluation.

The Database Access function enables that any data physically stored in any external repository can be used in policy conditions.

The benefits of this function are:

- Database access function hides to the application the data access protocol and the physical data model of the repository.
- Some of the SAPC data can physically be stored either in the SAPC internal database or in an external database. This allows deployments with different external databases.

Next figure shows an overview of the Database Access mechanism.

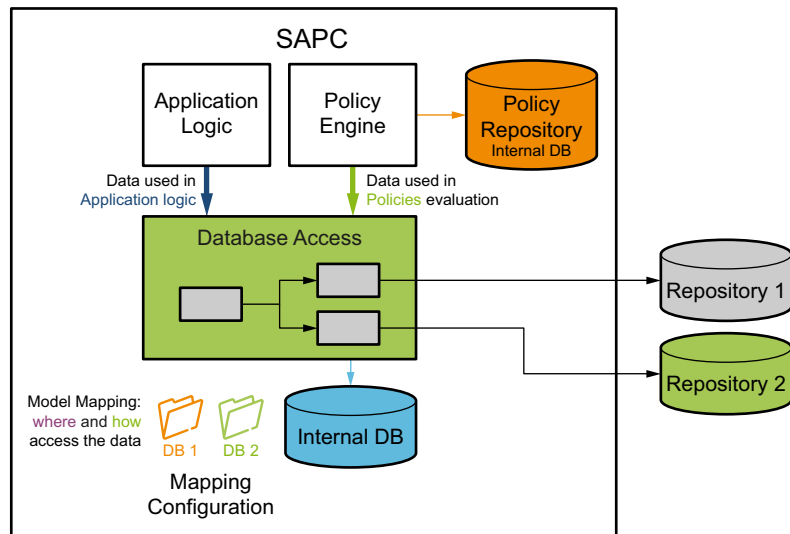


Figure 1 Database Access Mechanism Overview

This document uses the name **Entity** to refer to each of the SAPC application objects (logical objects) handled by Database Access that can be stored in an external repository. The **Entity** is also the configuration artifact that allows defining the model mapping between the application (internal) to repository (external) data model at runtime.

Next figure shows the main application objects in the SAPC.

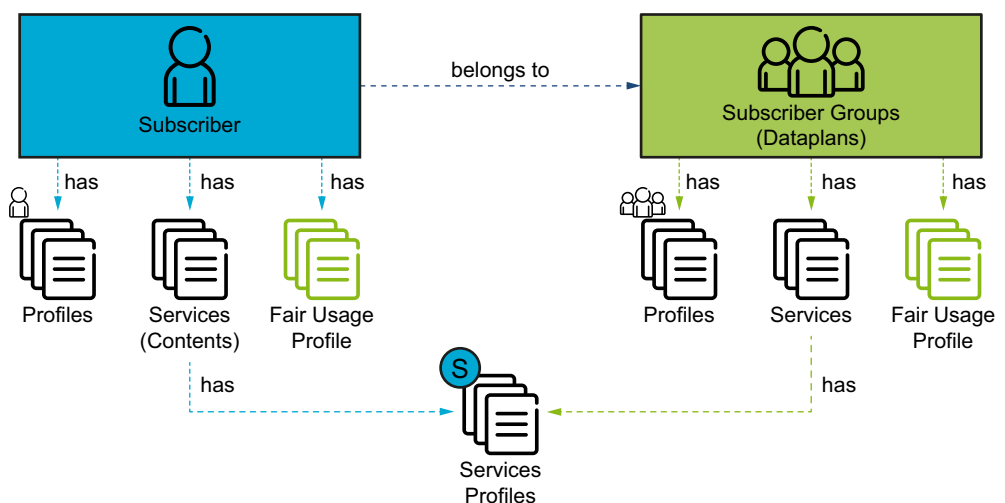


Figure 2 The SAPC Main Application objects

The application objects in the SAPC are predefined (provided at installation time) and cannot be divided in smaller elements.



Only the Subscriber Entity and related Entities can be mapped to an external repository data model.

2.2 Supported external databases Technologies

The SAPC supports the following external database access technologies:

- LDAP Server. For the optimal data model in an LDAP repository, see [Integration in User Data Consolidation](#).

2.2.1 Authenticated Connections to the external database

The SAPC supports user and password authentication when connecting to the external database. For details, see [Configure LDAP User Authentication](#) on page 22.

2.3 Data Model Mapping

To de-couple the SAPC application objects or policy data used in policy conditions and the physical database, the SAPC uses **Entity Data Source** concept: it is a representation of the mapping between application objects and the physical storage, that is, from where and how to access data.

Entity Data Sources contain the specification, of where the data is stored.

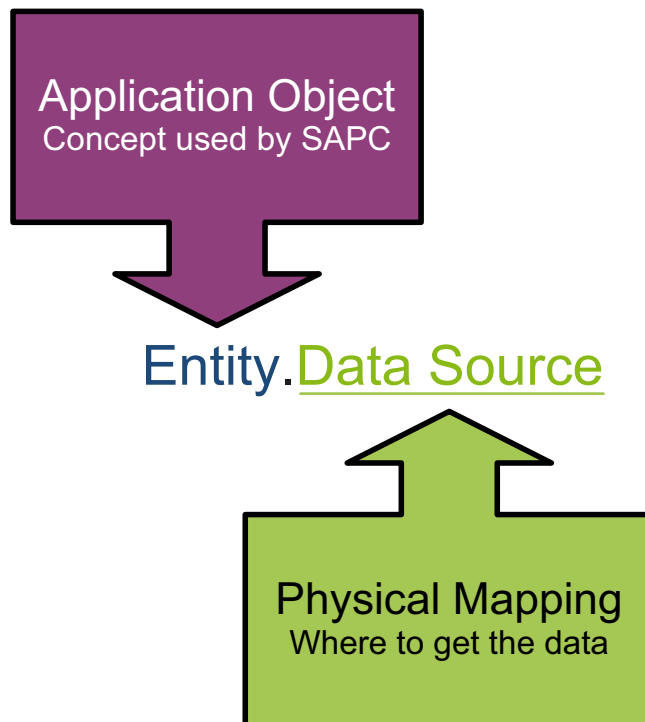


Figure 3 Entity Data Source Concept

Entity Data Source concept is reused not only when the physical data model is stored in an external database, but also when data is stored in the SAPC internal database (and this is the option provided at installation time).

An Entity Data Source definition is composed of a set of elements (using a language specified in [Entity Data Definition Language](#) on page 35) to define how to access the repository, where the data is stored, which attributes are obtained, and how to obtain them.

The main elements in an Entity Data Source are the following:

- A logical name for the object.
- The location information about the external database: URL, query, and some properties to make the search of the physical data.
- And the information about what attributes (fields) compose the application object, and how to obtain them. There can be following types of fields:
 - argument
 - constant
 - field



- reference to another application object: different application objects are related among them (for details, see [Entities Relations](#) on page 6).

Model Mapping

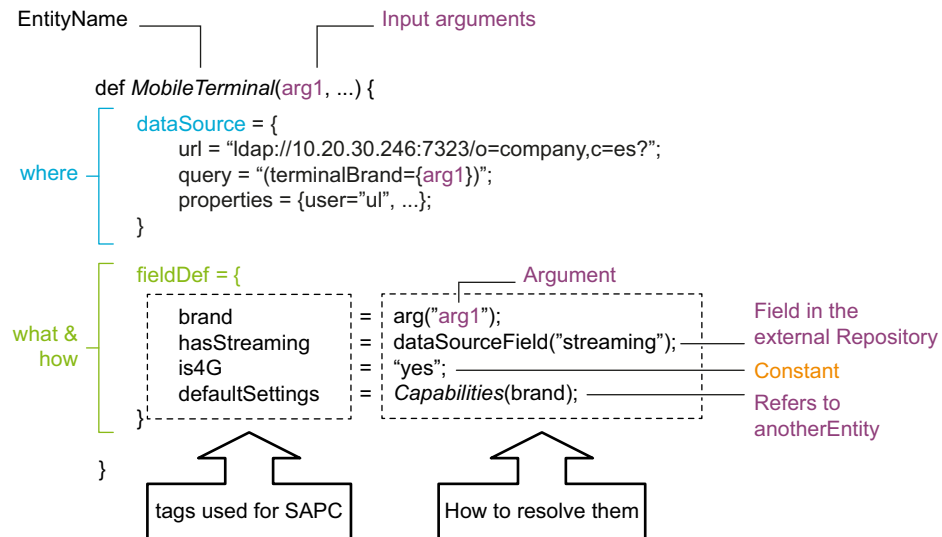


Figure 4 Entity Definition Main Parts

The mapping between the SAPC application data model and database model stored in external database is not forced to be a 1:1 relation. For example, an application object can be mapped to several physical objects. This allows the SAPC to be more flexible in adapting to different database models.

2.3.1 Entities Relations

Entities can relate ones to others in a chained way. There are two different ways of making relations among application objects:

- Structural: Entity Data Source can be nested, by using their fields, that is, an Entity Data Source field refer to another Entity Data Source. Next an example:

```
def Entity1( arg1) {
  dataSource = {
    url = ...;
  }
  fieldDef = {
    id = arg( "arg1" );
    field1 = dataSourceField("...");
    field2 = dataSourceField("...");
    entity2Data = Entity2( id );
  }
}
```



```

    }
}

def Entity2( arg2 ) {
  dataSource = {
    url = ...;
  }
  fieldDef = {
    id      = arg( "arg2" );
    fieldA  = dataSourceField("...");
    fieldB  = dataSourceField("...");
  }
}

```

The use of this case within a policy condition is as follows:
Entity1.entity2Data.fieldA.

These relationships (marked by the left sides of fieldDef block) are provided at installation time in the SAPC and the operator must not change them. The operator has the flexibility to define how to map to the physical actual data model (right sides using dataSourceField).

- Dynamic: For entities that are not predefined in advanced in the SAPC, their relationship can be established by configuration.

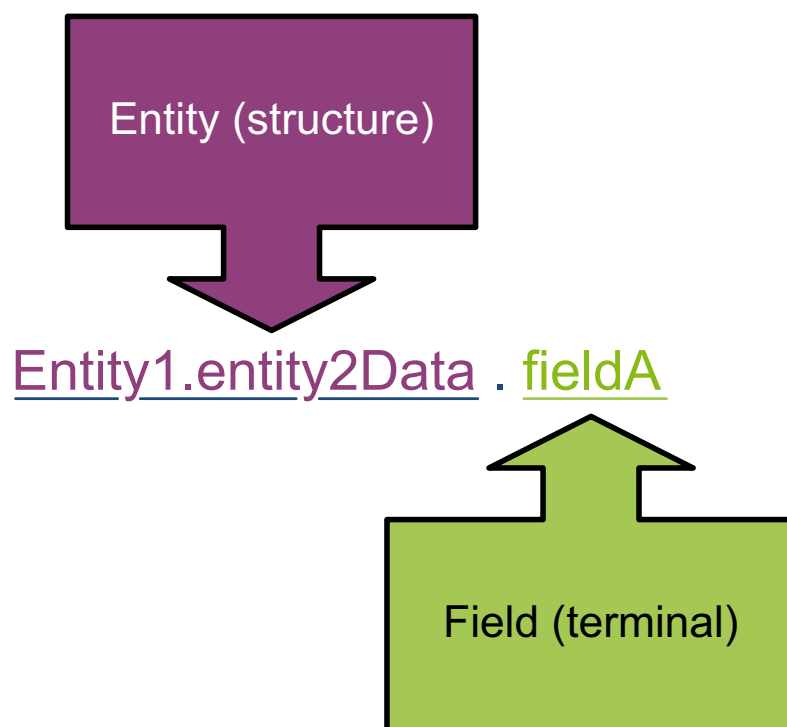


Figure 5 Chaining Entities

2.3.2 Entity Instance Searches

To perform a search, the SAPC supports single or composed keys. Entities can be linked based on a primary key (single input argument) scheme or based on composed key (multiple input arguments). The output result from a query done to locate an Entity, can be used as input parameter for another Entity search.

The SAPC can use as such keys data received from traffic messages (like MSISDN, UE IP Address, location, IMEI, APN) or obtained from another Entity. For more flexibility, the SAPC can also use policy functions (specified in Configuration Guide for Subscription and Policies) to transform these keys.

Next figure summarizes the relations among the main preconfigured Entities and other relevant application objects:

SAPC pre-installed EDSs: relations

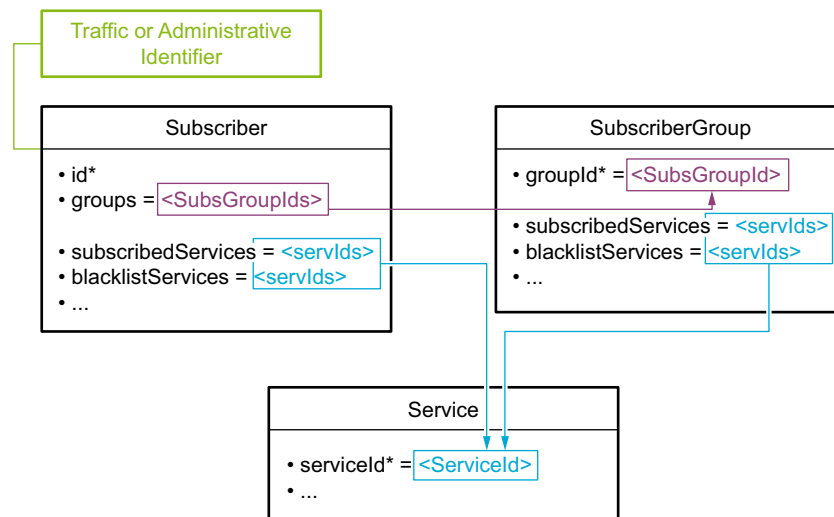


Figure 6 Subscriber, Subscriber Group and Service Relations

To process a traffic request corresponding for a Subscriber, the SAPC executes the following procedure:

1. The SAPC starts looking for the subscriber profile: for Subscriber (and its chained Entities), entries can be located used input arguments in two exclusive ways:
 - Single key: with a unique argument and no default value. The SAPC needs to consider an argument (key) for the subscriber. The SAPC uses the following procedure to determine the subscriber key:
 - a. The SAPC looks first if it is configured the SubscriberIdentity Entity Data Source, using as argument the value for the subscriber identifier received within the traffic request. If this is found, the



adminId field from that entity is used as the argument for the Subscriber entity.

- b. If SubscriberIdentity is not found, the SAPC looks for the Subscriber entity, using as argument the value received within the traffic request.

Note: The SAPC uses the following traffic identifiers:

- Subscription-Id AVP requests.

Example: `def Subscriber (argId)`

- Multiple keys: with several arguments.

Examples:

```
def Subscriber (arg1, arg2='AccessData.bearer.accessPoint')
```

2. The SAPC makes a search for each of the active groups of the subscriber. The SAPC business logic also uses an implicit "global" group for all subscribers. So, an extra query to fetch the "global" subscriber group data is done.
3. The SAPC makes a query for each of the services defined in the subscriber and active subscriber groups.

2.3.3

Use of Data to Evaluate Policy Conditions

The use of Entity Data Sources allows the SAPC to evaluate policy conditions using data not known in advance, and such data can be stored in an external database.

Note: The policy data (see Policy Repository in [Figure 1](#)) are always configured in the SAPC internal database.

Next figure explains the syntax of how new Entity attributes can be used inside policy conditions.

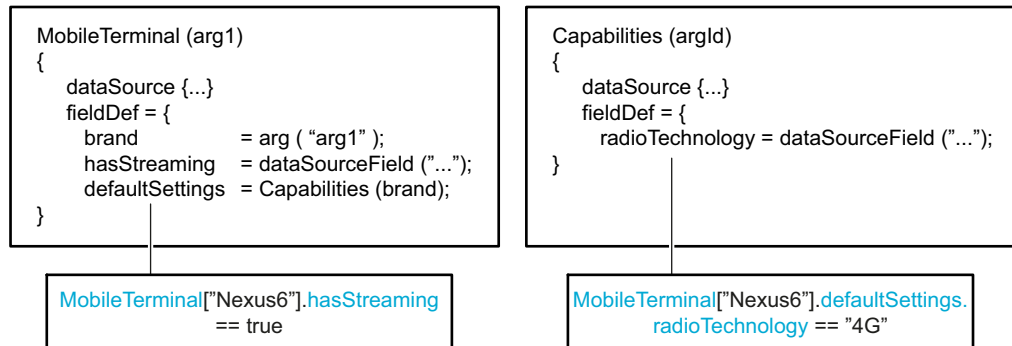


Figure 7 Example of Entity Tags to Be Used in a Policy Condition

2.4 Write Operations

Apart of read data from external databases, the SAPC supports to write Fair usage accumulators, ongoing session and closed session contexts on external LDAP repositories. For more information on ongoing session and closed session context writing operations, see [Session Context Exposure User Guide](#).

It is possible to read and write the Fair usage accumulators corresponding to the subscribers of a shared dataplan in a different location than usage accumulators for the individual subscribers.

2.5 Data Combinations Use Cases

This section explains which data combinations are possible and which are not.

2.5.1 Subscriber Data

Different pieces of the SAPC subscriber profile data can be located in different repositories, but with some considerations.

Only the subscriber data that are chained by chained Entities can be hosted in different repositories, given that each Entity Data Source can have a different data location defined.

Subscriber data extensions (for use in policy condition evaluation) through operator-specific information can only be located in the same repository where the subscriber is located. The advantage of this is that updates in the extended fields, trigger a reauthorization (through SOAP notification in case of LDAP access).

It is also possible to put subscriber data extension in a different repository, creating an entity for the subscriber. It is possible to refer directly to this entity or it is possible to chain it in the Subscriber entity. But updates in the data attribute



pointed by this configured entity does not always trigger a subscriber reauthorization.

2.5.1.1 Subscriber Data Extension in External Database

Some operators want to use their own extra data in the SAPC policy condition evaluation. In that case, the SAPC application objects can be extended with new fields. For example, extending the Subscriber with a new flagX that represents subscriber tariff plan.

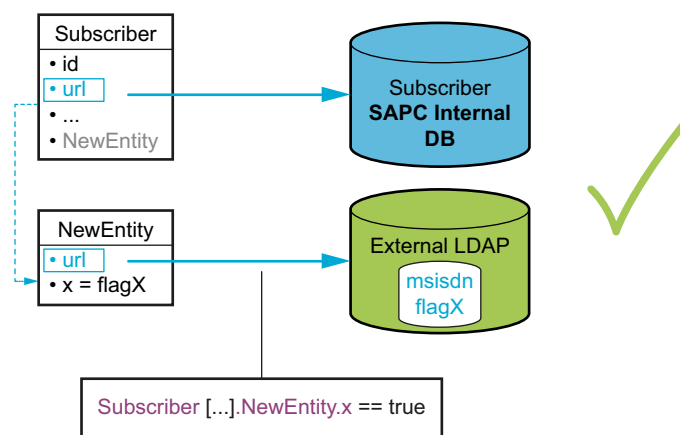


Figure 8 Subscriber Extension: New Data for Policy Evaluation

2.5.2 Data Extension in the SAPC internal database

It is possible to configure new data (not related to subscriber) being stored in the SAPC internal database for its use within policy conditions.

For example, the operator wants to check if a flag called flatRate is true, and if a request is being processed during promotion months. And the operator wants to store such extra data in the internal database provided by the SAPC.

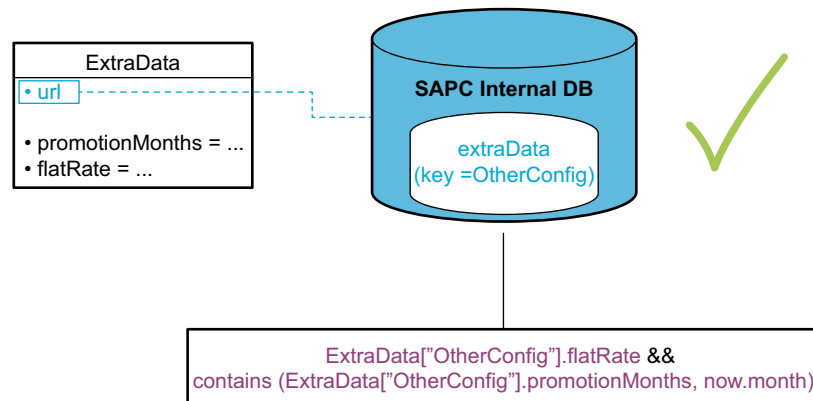


Figure 9 Subscriber Data Extension Stored in internal database

For the configuration details, see [Configure Extra Data In internal database](#) on page 24.

2.5.3 Distributing Same Data in Different Repositories

It is possible to distribute different entries of the same application object in different repositories. This is possible using multiple input arguments (see [Configure Keys for Application object Searches](#) on page 17). However, it is not possible to mix different database access technologies (for example some subscribers in LDAP and some other subscribers in internal database).

An example: some Subscribers are physically stored in LDAP Repository 1 or 2 depending on the country, according to [Figure 10](#).

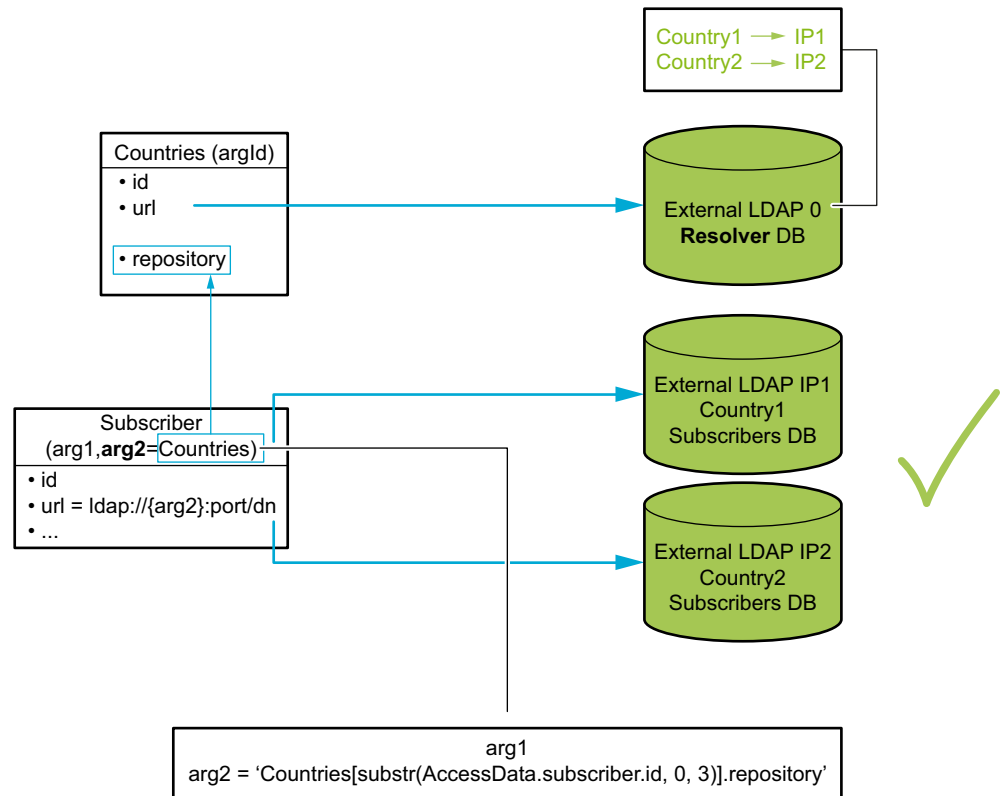


Figure 10 Subscribers Distributed By Country

To achieve this, it is needed to use several input parameters, and use a new Entity Data Source as extra argument to locate Subscribers that returns as output result the repository in which each subscriber is distributed. See [Example 6](#) for the configuration details.

2.5.4

Mixing Different Types of Repositories for An Application object

It is not possible to store some Subscribers in an external database (for example LDAP) and some others internally in the SAPC internal database. The reason is that the same Entity Data Source definition is used for both set of subscribers and its data location cannot point to different database access technologies.

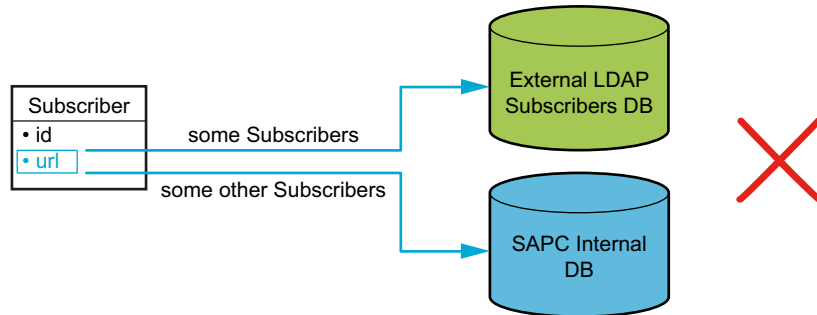


Figure 11 Not Possible to Mix Different Types of Repositories

2.5.5 Conditions on external database Data Model

The SAPC imposes a fixed application object data model, the one provided at installation time.

Furthermore, for external LDAP, Ericsson recommends as the optimal physical database schema (fully verified), the one for UDC (see [Integration in User Data Consolidation](#)). There are some constraints in mentioned schema. For example, Subscriber groupId field must be multivalued.

Not Needed Data

When an application object related to a particular function is not used, it is not needed to complete all its fieldDefs; it is possible to modify the SAPC preconfigured Entity Data Source, to leave empty the right side of the fieldDef.

As example, next figure shows how to skip end-user notifications and fair usage data:

Subscriber	
<pre>fieldDef = { id = dataSourceField("id"); groups = ...; subscribedServices = ...; blacklistServices = ...; usageLimits = " "; sms = " "; }</pre>	

Figure 12 Shortcut for not Needed Data



2.6 Performance Optimizations

This section includes relevant information to achieve an optimal performance when the SAPC fetches data from an external database.

It is possible to adjust in the SAPC time-outs related to the connections and queries done to the external database (see details in [Table 3](#)).

2.6.1 LDAP Repositories

For the optimal physical data model in an LDAP repository, read [Integration in User Data Consolidation](#): it contains the LDAP data model (DIT and schema) for Subscriber profile data, and the corresponding Entity Data Sources configuration. It also explains error, connection handling, database failover, and failback mechanisms that the SAPC supports with LDAP.

Ericsson recommends that the DIT schema has the less number of levels as possible. It is also recommended to gather under the same object class, as many data as possible representing Subscriber profile.

To reduce the number of accesses and retrieve several entries at once, it is possible to use some standard LDAP features such as **scope**, **filtering**, and **alias de-referencing**. Its use depends on the support of such functions in the external LDAP repository and on the particular operator data model, see *Lightweight Directory Access Protocol, RFC 4511*.

Next figure shows the different LDAP scopes.

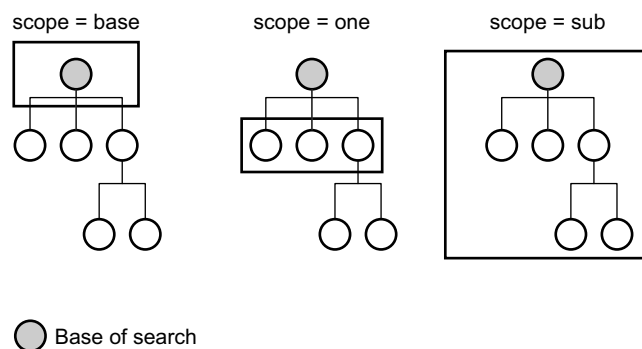


Figure 13 LDAP Scopes

To minimize the number of searches, multiple object entries can be fetched using the same LDAP query. To do that:

- Use subtree scope, to get all object classes under a particular DIT node.



- Use **alias** to obtain as well the objects stored in a different DIT branch. The use of alias implies some extra provisioning in the external LDAP repository (by using alias (never/always/search/find) and extensibleObject object classes).

Warning!

When scope subtree is used and aliases dereferencing is indicated, the first ldapsearch request can be slower (several entries are retrieved), but the rest of the requests are faster as the data are cached.

2.7 Fallback Mechanism for Subscribers in External Databases

It is possible to provision in the SAPC special unknown subscriber residing in a different repository than rest of subscribers. This is useful in case the external database fails: then, subscribers information cannot be retrieved. Instead of rejecting the requests, the SAPC can process traffic using the characteristics of unknown subscriber entry. This enhances availability in the network.

When the SAPC detects that the external database fails, it looks for the "SubscriberUnknown" Entity Data Source, and obtains the especial unknown subscriber entry.

Note: For unknown subscriber, the explanation in [page 8](#) does not apply.



3 Database Access Operation and Maintenance

3.1 Configure Entity Data Sources

Entity Data Sources are configured inside the SAPC internal database, using COM class `EDSource` MOC.

The SAPC provides at installation time the configuration for application objects pointing to internal database.

Warning!

To work with the SAPC using its internal database, do not modify the `class EDSource` instances provided at installation time. Otherwise it can lead to errors in getting the application object data, or in a performance penalty.

3.1.1 Configure Keys for Application object Searches

It is possible to use one or several input arguments (keys) to support composed indexes when retrieving Entities from external database in a flexible way. These arguments are the variable parts that can be used to perform the query towards the external database.

The input arguments are the first thing specified in the Entity Data Source, in definition attribute, between parenthesis before the `dataSource` block.

Default values for the input arguments can be specified (in the same way as in a programming language) using `'<default_value>'` notation.

It is possible to use as default value any of the supported policy expressions according to the policy language specified in [Configuration Guide for Subscription and Policies](#).

Some examples of input arguments are:

```
'AccessData.subscriber.id'  
'AccessData.subscriber.ueIpAddress'  
'AccessData.subscriber.locationInfo.sgsnAddress'  
'AccessData.userEquipmentInfo.model'  
'AccessData.bearer.accessPoint'
```

For newly defined Entities, default values can be used in every argument.



Warning!

Do not introduce infinite loops in case an argument of Entity1 makes reference to an attribute of the Entity1 itself or references another Entity2 that directly or indirectly uses any Entity1 field.

The following example shows what to avoid:

```
def Subscriber ( arg1='AccessData.subscriber.id',
                arg2 = 'Subscriber.birthDate' ){
  dataSource = { ... }
  fieldDef = {
    id = dataSourceField("id");
    groups = dataSourceField("dataplan");
    trafficIds = dataSourceField("trafficIds");
    sharedDataplan = dataSourceField("sharedDataplanId");
    subscribedServices = dataSourceField("subscribedContents");
    blacklistServices = dataSourceField("deniedContents");
    ...
    activationDate = dataSourceField("name:subscriptionDate");
    cellID = dataSourceField("name:cellId");
    birthDate = dataSourceField("name:birth");
  }
}
```

Previous example shows that default argument for arg2 in Subscriber Entity makes a reference to birthDate attribute of Subscriber Entity itself.

3.1.1.1 Use of Arguments in Policy Conditions

Within policy conditions, it is possible to use Entities with or without arguments. By specifying default arguments in an Entity definition, the use of the Entity fields in policy conditions can be simplified. The arguments not specified in the policy condition are automatically resolved from the default arguments.

Consider the following example:

```
def Entity1( arg1='AccessData.subscriber.id' ) {
  dataSource = {
    url = ...;
  }
  fieldDef = {
    id = arg( "arg1" );
    field1 = dataSourceField("...");
    field2 = dataSourceField("...");
  }
}
```

For the sake of easiness, it is possible to use within a policy condition the following:

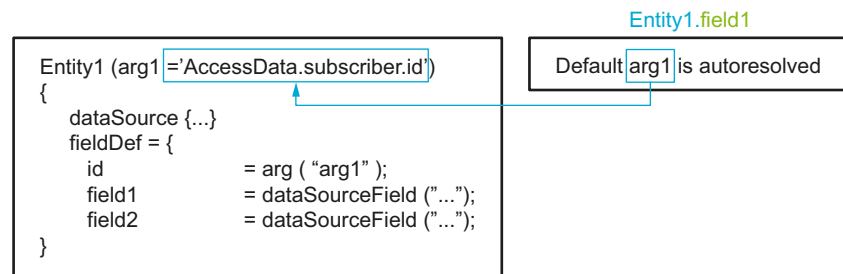


Figure 14 Default Argument Auto Resolution

And now, consider following extension of the example:

```
def Entity1( arg1) {
  dataSource = {
    url = ...;
  }
  fieldDef = {
    id          = arg( "arg1" );
    field1      = dataSourceField( "...");
    field2      = dataSourceField( "...");
    entity2Data = Entity2( id );
  }
}

def Entity2( arg2, arg3 = 'AccessData.bearer.accessPoint' ) {
  dataSource = {
    url = ...;
  }
  fieldDef = {
    id          = arg( "arg2" );
    fieldA      = dataSourceField( "...");
    fieldB      = dataSourceField( "...");
  }
}
```

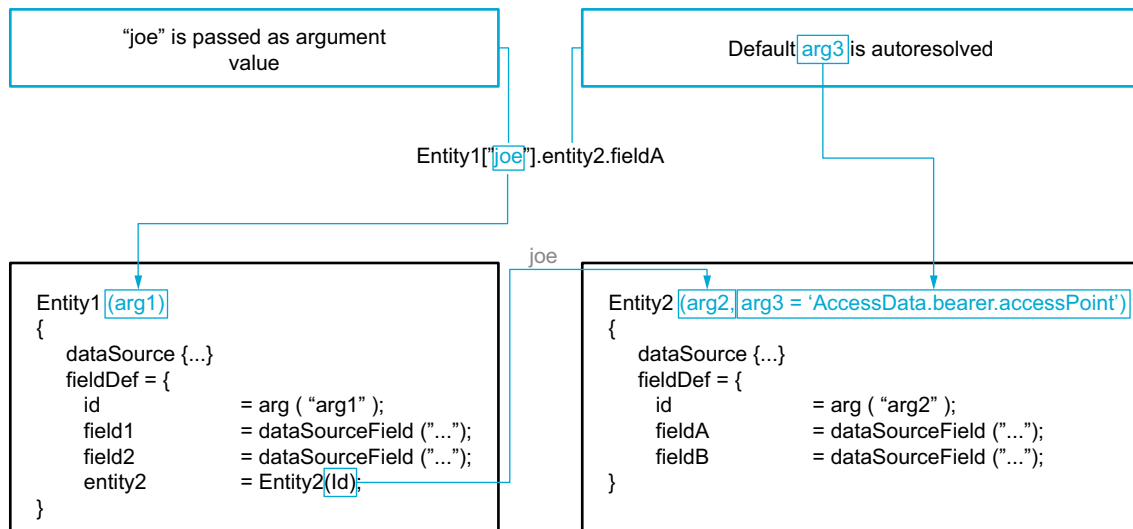


Figure 15 Multiple Input Arguments and Default Value

3.1.2 Data Restrictions

Entity Data Sources provided at installation time cannot be deleted. They can be modified, but the following restrictions apply to their content:



SAPC pre-installed EDS: Example

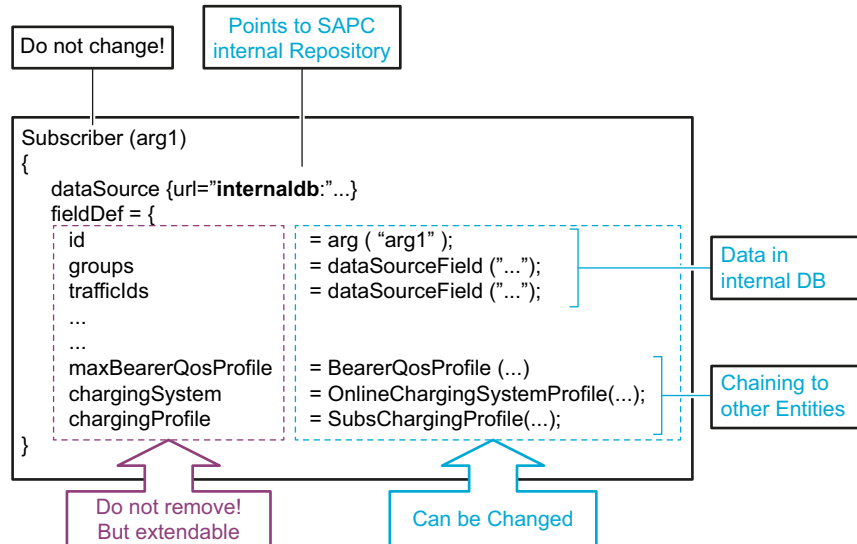


Figure 16 Example of Preconfigured Subscriber Entity Data Source

The complete list of fieldDef attributes for an Entity, can be get by COM, getting the corresponding EDSsourceId instance.

- The eDSsourceId value cannot be modified.
- The Entity Data Source name in the definition attribute cannot be modified.
- The field names (left sides) defined within the fieldDef section in the definition attribute cannot be changed neither removed. The right sides can be modified, with the consideration that they have to be of the same type.

Do!

The names of the LDAP attributes used in the right sides of the fieldDef section, are case-sensitive, they have to match exactly with the external LDAP schema.



3.1.3 Configure to Write

Caution!

Do not remove the LDAP attribute (physical data model) where usage accumulator are stored in the external database: this can seriously affect the Fair Usage control function. Note also, that the SAPC by now only supports write operations of Subscriber usage accumulators in an LDAP repository.

To write Subscriber usage accumulators in an LDAP repository, but Family usage accumulators in internal database, define AccumulatedUsage Entity Data Source pointing to LDAP, and AccumulatedUsageSharedDataPlan to internal database. For configuration details, see [Example 8](#).

3.1.4 Configure Fallback for Subscribers

To apply this fallback, follow the steps explained for unknown subscriber in [Configuration Guide for Subscription and Policies](#), with the following exception: leave the usageLimits field empty using empty string "".

Assure that SubscriberUnknown and GroupsToSubscriberUnknown Entity Data Sources point to the SAPC internal database (url="internaldb:").

3.2 Configure LDAP User Authentication

To use user and password authentication (that is, not anonymous LDAP sessions) for the connection to the LDAP external database:

Steps

1. Set "user" property within dataSource block in the Entity Data Source definition.
2. Set the password of the LDAP repository in dbPassword attribute in EDSOURCE object.

Next table shows how the SAPC does the authentication against the LDAP Server, depending on what it is configured in the corresponding EDSOURCE object:

Table 1 LDAP Authentication Options in the SAPC

LDAP Authentication in the LDAP Server	Property "user" configured?	Attribute dbPassword configured?
Anonymous	No	No
Anonymous	No	Yes



LDAP Authentication in the LDAP Server	Property "user" configured?	Attribute dbPassword configured?
LDAP bind using user but empty password ⁽¹⁾	Yes	No
LDAP bind using user and password credentials	Yes	Yes

(1) This leads to receive invalid credentials error (49) in the LDAP Server.

For a configuration example, see [Example 6](#).

3.3 Configure Use of LDAP Scope, Filter, Alias Dereferencing

To perform LDAP search using scope and filter, use an extension subblock delimited by curly brackets "{}" within the query element configured in the Entity Data Source definition, according to following syntax:

- Filters to return a single entry:

```
query="(<ldap_filter>);"
```

- Or, multiple entries returned:

```
query = "? scope=sub ? ( <ldap_filter> ) ?
{ alias=always;
  entry= ou={arg1},serv=SAPC,mcsid={arg2};;}";
```

entry has to match the actual DN of the LDAP entry returned by the ldapsearch.

To better understanding, consider the following analogy: a graphical LDAP browser, that shows the tree of several LDAP entries; the SAPC can retrieve all the entries (as the LDAP browser does), but the content (attributes) of each instance is not shown, until the particular entry is clicked. entry is equivalent to that "click". The ldapsearch can return several entries, but it is what stated in configuration of the entry subblock what makes possible the final access to their attributes.

"?" symbol is the separator between scope, filter, and rest of data. Its use at the beginning and end of the query entry is optional or even removed when only filter is used. When specifying "?", it is mandatory to specify also scope.

See scope options supported by the SAPC in [Figure 13](#).

When scope is not explicitly configured, the SAPC assumes subtree.

When alias is not specified, default value is always.



When scope=sub or scope=one are used, then use of entry is mandatory.

Do!

Before configure these options in the Entity Data Source, verify the external LDAP repository capabilities, to check that it supports aliases dereferencing and search based on subtree.

3.4 Database Access Configuration Examples

3.4.1 Using Functions to Modify the Subscriber Identifier

Next example illustrates the case where before retrieving a Subscriber instance from an LDAP repository, the argument (MSISDN) is transformed by substituting the two first characters by '06'. If the original argument was 34701234567890, the transformed one is 06701234567890, and the last one is the used to retrieve the Subscriber data from the repository.

```
def Subscriber( argId =
    'strcat("06", substr((AccesData.subscriber.id), 2, 0))' ) {
    dataSource = {
        url = "ldap://10.20.30.40:7323/o=company,c=es?";
        query = "(msisdn={argId})";
    }
    fieldDef = {
        ...
    }
}
```

3.4.2 Configure Extra Data In internal database

To evaluate new data in policy conditions, but physical store their values in the SAPC internal database, do the following:

Steps

1. Create the extra data in the SAPC, using operator-specific-infos URI in the provisioning REST API.
2. Create the corresponding instance of EDSource, pointing to the newly created data.

[Example 1](#), [Example 2](#) illustrate an example.



Example 1 Extra data in the SAPC Internal Database

PUT /operator-specific-infos/OtherConfig

```
{
  "infoList" :
  [
    {
      "attributeName" : "flatTariff",
      "attributeValue" : "yes"
    },
    {
      "attributeName" : "promoMonths",
      "attributeValue" : "7"
    },
    {
      "attributeName" : "promoMonths",
      "attributeValue" : "8"
    },
    {
      "attributeName" : "promoMonths",
      "attributeValue" : "9"
    }
  ],
  "infoId" : "OtherConfig"
}
```

Example 2 Extra data in the SAPC internal database

```
<edit-config>
  <target>
    <running />
  </target>
</edit-config>
<config>
  <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
    <managedElementId>1</managedElementId>
    <dnPrefix>dc=ManagedElement</dnPrefix>
    <networkManagedElementId>1</networkManagedElementId>
    <userLabel>Managed Element</userLabel>
    <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
      <policyControlFunctionId>1</policyControlFunctionId>
      <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
        <entityDataId>1</entityDataId>
        <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom">
          <eDSourcesId>1</eDSourcesId>
          <EDSource xmlns="urn:com:ericsson:ecim:edsourcesmom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="merge">
```



```

                                <eDSourceId>ExtraData</eDSourceId>
                                <definition>
def ExtraData( argId ) {
dataSource = {
    url = "internaldb:";
    query = "OperatorSpecificInfoPot:{argId}";
}
fieldDef = {
    flatRate = dataSourceField("name:flatTariff");
    promotionMonths = dataSourceField("name:promoMonths");
}
}
                                </definition>
                                </EDSource>
                                </EDSources>
                                </EntityData>
</PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>

```

This example creates "OtherConfig" REST resource, containing attributes "flatTariff" (with value yes) and "promoMonths" (with values 7, 8 and 9). Then, ExtraData["OtherConfig"].flatRate and ExtraData["OtherConfig"].promotionMonths are ready to be evaluated in policy conditions.

3.4.3 LDAP Configuration Examples

For the detailed configuration to hold application Subscriber objects in an external LDAP server (CUDb), the optimal data model can be found in Integration in User Data Consolidation.

The rest of examples provided in this section, show configuration examples for a generic LDAP server, for the shake of learning.

3.4.3.1 New Subscriber Data in an External LDAP Repository

[Example 3](#) is an example of personal connection information (that can reside in an external LDAP server), extending the SAPC Subscriber data, so that the SAPC can evaluate this information in policy conditions:

The definition for the preferredConnectionTimes Entity Data Source could be the following:

Example 3 preferredConnectionTimes Entity Stored in an External LDAP Server

```

<edit-config>
  <target>
  <running />

```



```

</target>
<config>
<ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
  <managedElementId>1</managedElementId>
  <dnPrefix>dc=ManagedElement</dnPrefix>
  <networkManagedElementId>1</networkManagedElementId>
  <userLabel>Managed Element</userLabel>
  <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
    <policyControlFunctionId>1</policyControlFunctionId>
    <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
      <entityDataId>1</entityDataId>
      <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
    >
      <eDSourcesId>1</eDSourcesId>
      <EDSource xmlns="urn:com:ericsson:ecim:edsource →
mom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operat →
ion="merge">
        <eDSOURCEId>preferredConnectionTimes</eDSour →
ceId>
        <definition>
          def preferredConnectionTimes ( msisdArg )
          {
            dataSource = {
              url = "ldap://99.66.55.44:2121/o=company,c=es?";
              query = "(msisdn={msisdArg})";
            }
            fieldDef = {
              lastConnectionTime =
                dataSourceField( "lastConnTime" );
              weekAverageConnectionTime =
                dataSourceField( "weekAvgConnTime" );
            }
          }
        </definition>
      </EDSource>
    </EDSources>
  </EntityData>
</PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>

```

This example uses an anonymous LDAP connection to LDAP Server on IP 99.66.55.44 on port 2121. The fields in the LDAP server, under the entry keyed by "msisdn", are "lastConnectionTime" and "weekAverageConnectionTime".



3.4.3.2 New Entity Stored in an External LDAP Repository

Example 4 is an example of an Entity Data Source definition for a new (not known in advanced in the SAPC) data stored in an external database containing mobile terminal capabilities.

Example 4 MobileTerminalCapabilities Stored in an External LDAP Server

```
<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
        >
          <eDSourcesId>1</eDSourcesId>
          <EDSource xmlns="urn:com:ericsson:ecim:edsource →
mom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operat →
ion="merge">
            <eDSourceId>MobileTerminalCapabilities</eDSou →
urceId>
              <definition>
                def MobileTerminalCapabilities( argBrand ) {
                  dataSource = {
                    url = "ldap://10.20.30.246:389/o=company,c=es?";
                    query = "(terminalBrand={argBrand})";
                  }
                  fieldDef = {
                    is4g = dataSourceField( "4gEnabled" );
                    displayResolution = dataSourceField( "screen" );
                    hasStreaming = dataSourceField( "streamingEnabled" );
                  }
                }
              </definition>
            </EDSource>
          </EDSources>
        </EntityData>
      </PolicyControlFunction>
    </ManagedElement>
  </config>
</edit-config>
```




In this example, `MobileTerminalCapabilities` entries are retrieved from an external LDAP server. The terminal brand is used as argument to locate the objects.

The `is4g` field is obtained from the `4gEnabled` attribute in the LDAP server. The `displayResolution` field is obtained from the `screen` attribute in the LDAP server. And the `hasStreaming` field is obtained from the `streamingEnabled` attribute in the LDAP server.

3.4.3.3

Subscriber Distributed across Several LDAP Repositories

It is possible to distribute Subscribers among different repositories, for example based on a geographical distribution according to the Subscriber identity.

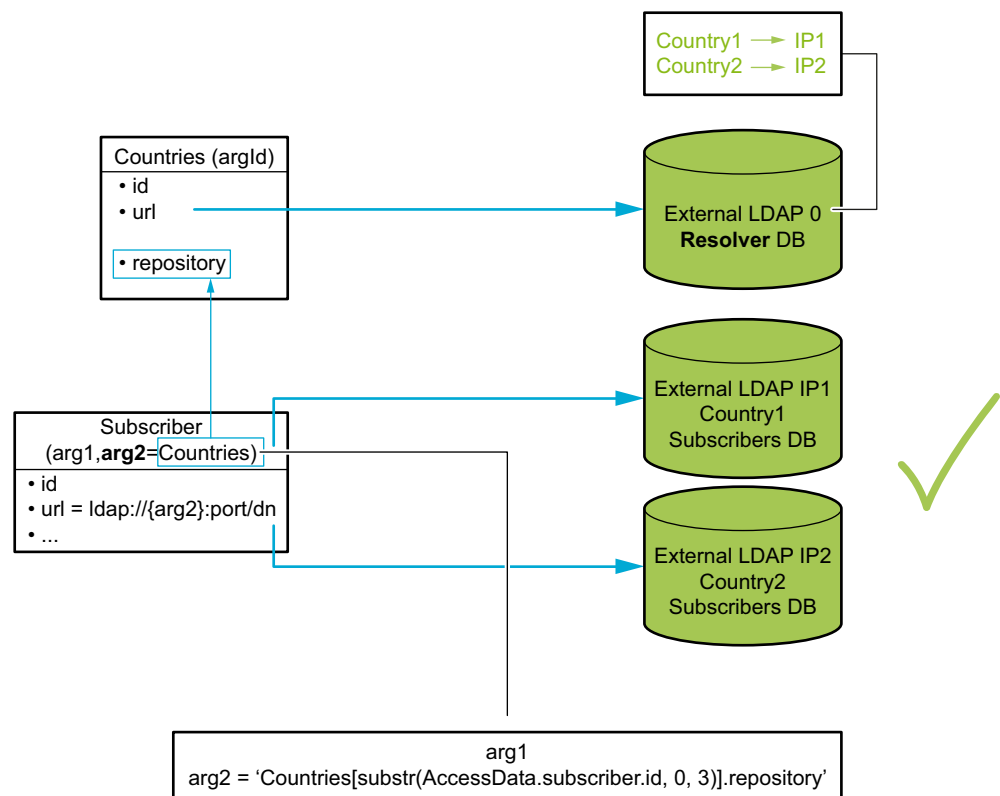


Figure 17 Distributed Subscribers

[Example 5](#), [Example 6](#) illustrate a case where Subscribers are distributed depending on their country: the country is extracted from the first 3 digits of their MSISDN (using `substr` function), and the `Countries` Entity Data Source returns the IP Address of the external database to use in the Subscribers queries.



Note: Similarly, the SAPC supports Subscriber distribution based on the APN (use `AccessData.bearer.accessPoint` policy tag).

First of all, an Entity Data Source definition for a new entity to identify different countries repositories. Then the Subscriber Entity Data Source definition is modified:

Example 5 Repositories by country

```
<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
        >
          <eDSourcesId>1</eDSourcesId>
          <EDSource xmlns="urn:com:ericsson:ecim:edsourcesmom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="merge">
            <eDSourcesId>Countries</eDSourcesId>
            <definition>
def Countries ( argId )
{
  dataSource = {
    url = "ldap://10.1.20.21:7323/o=company,dc=com?";
    query = "(countryCode={argId})";
    properties = {user = "user1";};
  }
  fieldDef = {
    id = arg( "argId" );
    repository = dataSourceField("ldapServerIp");
  }
}

            </definition>
            <dbPassword>
              <cleartext></cleartext>
              <password>passwd_ldap1</password>
            </dbPassword>
          </EDSource>
        </EDSources>
      </EntityData>
    </PolicyControlFunction>
```



```

    </ManagedElement>
  </config>
</edit-config>

```

Example 6 Subscribers distributed in different Repositories

```

<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
            >
              <eDSourcesId>1</eDSourcesId>
              <EDSource xmlns="urn:com:ericsson:ecim:edsourcem →
om" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operati →
on="merge">
                <eDSourcesId>Subscriber</eDSourcesId>
                <definition>
                  def Subscriber
                    ( arg1,
                      arg2 = 'Countries[substr(AccessData.subscriber.id, 0, 3)]. →
repository' )
                    {
                      dataSource = {
                        url = "ldap://{arg2}:7323/ou=subscribers,o=company,dc=com →
,?";
                        query = "(msisdn={arg1})";
                        properties = {user = "user2";};
                      }
                      fieldDef = {
                        ...
                      }
                    }
                  </definition>
                  <dbPassword>
                    <cleartext></cleartext>
                    <password>passwd_ldap2</password>
                  </dbPassword>
                </EDSource>
              </EDSources>
            </EntityData>
          </EntityData>
        </EntityData>
      </PolicyControlFunction>
    </ManagedElement>
  </config>
</edit-config>

```



```

    </PolicyControlFunction>
  </ManagedElement>
</config>
</edit-config>

```

Note: This example focus on how to achieve the subscriber distribution using input arguments. It does not intentionally show the detailed fieldDefs of Subscriber Entity. To be a valid configuration, please complete it using the fieldDefs of the preinstalled eDSourceId=Subscriber.

3.4.4 Configure New Entity with Constants

[Example 7](#) is an Entity Data Source definition for a new entity that only contains constants:

Example 7 NodeConstants Entity

```

<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
            <eDSourcesId>1</eDSourcesId>
            <EDSource xmlns="urn:com:ericsson:ecim:edsource →
mom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operat →
ion="merge">
              <eDSourceId>NodeConstants</eDSourceId>
              <definition>
                def NodeConstants ( )
                {
                  dataSource = { url = ""; query = ""; }
                  fieldDef = {
                    countryCode = "34";
                    maxBitRate = "8000";
                  }
                }
              </definition>
            </EDSource>
          </EDSources>
        </EntityData>
      </PolicyControlFunction>
    </ManagedElement>
  </config>
</edit-config>

```



```

    </PolicyControlFunction>
  </ManagedElement>
</config>
</edit-config>

```

In this example, NodeConstants is an entity that is not retrieved from any repository, as the values for its fields are included inside the field definitions. This type of entities is useful to define constants for values that are used in several policy conditions.

3.4.5 Configure Subscriber Data Extension In the SAPC internal database

See **Configure Subscriber Data Extension** in Configuration Guide for Subscription and Policies.

3.4.6 Configure Fair usage accumulators for Read and Write

[Example 8](#) shows how to configure Entity Data Sources to read and write usage accumulators related to subscribers from an external LDAP, but usage accumulators for a shared dataplan in the SAPC internal database.

Example 8 Subscriber accumulators

```

<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
            <eDSourcesId>1</eDSourcesId>
            <EDSource xmlns="urn:com:ericsson:ecim:edsource →
mom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operat →
ion="merge">
              <eDSourceId>AccumulatedUsage</eDSourceId>
              <definition>
                def AccumulatedUsage ( argId ) {
                  dataSource = {
                    url = "ldap://10.20.30.40:389/
                      accumId={argId},

```



```

        ou=SAPC,mscId={argId},ou=multiServiceConsumer,
        ou=Customer_DB,dc=myDomain,dc=com";
    query = "?scope=sub?(objectclass = *)?{alias=always;entry →
=serv=PCRF,
        mscId={argId},ou=multiServiceConsumer,
        ou=Customer_DB,dc=myDomain,dc=com;}";
    }
    fieldDef = {
        id = dataSourceField("accumId");
        data = dataSourceField("accumData");
    }
}
</definition>
</EDSource>
</EDSources>
</EntityData>
</PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>

```

3.4.7 Configure Session Context for Read and Write

For the detailed configuration to read and write session context from an external LDAP, see [Integration in User Data Consolidation](#).

3.4.8 Examples of Policy Conditions Using Entity Data Source

This section contains some examples on how the tags defined in Entity Data Source can be used within policy conditions.

3.4.8.1 Example Using Data Expanding Subscriber

To evaluate conditions using the `LastConnTime` attribute of the PcT shown in [Example 3](#), use the following policy tag:

```
Subscriber.preferredConnectionTimes.lastConnectionTime.
```

It is not needed to specify the `subscriberId`, as it is automatically passed to the Subscriber Entity Data Source when the condition is evaluated.

3.4.8.2 Example Using New Entity

To access the `is4g` field for a mobile terminal shown in [Example 4](#), use the following policy tag (the terminal brand parameter could be obtained, for example, from the subscriber entity if available):

```
MobileTerminalCapabilities[<terminal brand>].is4g.
```



In case it is defined a new Entity (not chained from others):

```
def NewEntity (argId) {
  dataSource = { ... };
  fieldDef = {
    field1 = dataSourceField("")
  };
}
```

Assuming that this NewEntity is keyed using the subscriber identifier, it can be used in this way: `NewEntity[AccessData.subscriber.id].field1`

3.4.8.3 Example Using a Constant

To access the countryCode constant shown in [Example 7](#), use the following policy tag:

```
NodeConstants.countryCode
```

3.5 Entity Data Definition Language

The language for the definition of an Entity Data according to Backus-Naur Form (BNF) specification has the following format:

```
<entityDataDefinition> ::=
  "def" <entityDataName> [ "(" <argList> ")" ]
  "{"
    "dataSource" "=" "{" <dataSourceLocation> "}"
    "dataTarget" "=" "{" <dataTargetLocation> "}"
    "transformation" "=" "{" <transformDef> "+" "}"
    "fieldDef" "=" "{" <field>+ "}"
  "}"
<argList> ::=
  <argName> [ "=" "'<default_value>'" ] |
  <argName> ", " <argList>
<dataSourceLocation> ::=
  "url" "=" <urlExpression>
  [ "query" "=" [ "{parent}/" ] <queryExpression> ]
  [ "properties" "=" "{" <property>+ "}" ]
<dataTargetLocation> ::=
  "url" "=" <urlExpression>
  [ "properties" "=" "{" <property>+ "}" ]
<property> ::=
  <propertyName> "=" <propertyValue> "; "
<transformDef> ::= <fieldName> "=" "'<transformFunction>'"
<field> ::=
```



```

<fieldName> "=" <constant> ";" |
<fieldName> "=" <argument> ";" |
<fieldName> "=" <dataSourceField> ";" |
<fieldName> "=" <finalField> "+" <finalField> ";" |
<fieldName> "=" <dataSourceRef> ";" |
<fieldName> "=" "arrayOf" "(" <dataSourceRefs> ")" ";"

```

```

<finalField> ::= <constant> | <argument> | <dataSourceField>

```

```

<argument> ::= "arg(" <argName> ")"

```

```

<dataSourceField> ::= "dataSourceField" "(" <attrName> ")"

```

```

<dataSourceRefs> ::= =
    <dataSourceRef> |
    <dataSourceRef> "," <dataSourceRefs>

```

```

<dataSourceRef> ::= =
    <dataSourceNameRef> "(" <refArglist> ")"

```

The meaning of the definition language elements is the following:

Table 2 Entity Data Definition Language Main Element

Element	Description	Comments
<entityDataName>	Name of the application object whose data is stored in one repository or distributed across several repositories.	This name is a string used to reference the application object (together with the field names separated by dots) inside policy conditions (only for Entity Data Source)
<argList>	List of arguments passed to the application object.	Typically used to find the application object in the repository, but not only. For example, the subscriberId for the Subscriber.
<dataSourceLocation>	Contains information about the repository where the application object is stored and how to obtain the data from it.	
url	Specifies the location of the repository where the entity is stored with an <urlExpression>	—For the SAPC internal database: "internaldb:" —For LDAP: it is an LDAP URL: "ldap:<url>"



Element	Description	Comments
		It can be empty for application object containing only constant fields, or fields referencing another application object.
query	Specifies a query on the repository to obtain an instance of an application object with a <code><queryExpression></code> . The query depends on the type of repository. Arguments can be referenced inside a query by enclosing them between "{ }".	—For LDAP: it corresponds with an LDAP filter. It is optional. When not specified, the SAPC assumes filter (objectclass=*).
<code><property></code>	Represents a property needed for the access to the external database, depending on the particular database technology. See Table 3 .	
field	For every ,Entity Data Source each field contains a name of a field (<code><fieldName></code>) to be used in policy engine conditions to reference the field and an expression on how to obtain the field information. See Table 4	
transformFunction	One of the functions that can be used within condition formula. Refer to Configuration Guide for Subscription and Policies . Transformation is supported for fields passed as argument (<code><argList></code>) to the entities. Once the transformation is indicated, the fields always work considering their modified values. That means that if a transformation is used, the change is done before the query to find a concrete object instance in the repository is composed.	

Table 3 Properties for LDAP Access

Property	Description	Default Value	Comments
user	User used to create LDAP sessions (not anonymous)	-	user = "<user value>"



Property	Description	Default Value	Comments
ConnTimeout	Controls the time that the SAPC waits until a connection is established. It affects only queries where it is needed to create a connection towards an external database.	4000 ms.	Increasing this time-out is indicated whenever slow connections are expected. With a higher value, in case there is a failure in the connection, the SAPC answers the traffic requests more slowly, which could have an important impact in performance.
QueryTimeout	Once the connection is got, it controls the time that the SAPC waits until the result is returned after the query. It affects all traffic queries.	2000 ms.	

Table 4 Fields

Field Type	Description	Example
constant	It is a literal value, that is either a string enclosed by double quotes or a number.	<code>countryCode = 34</code>
Input argument	It is a reference to an argument passed to the Entity Data Source using the <code>arg(<argName>)</code> expression.	<code>id = arg("argId");</code>
Entity Data Source reference:	<p>Contains the name of another Entity Data Source where the information of the field is stored.</p> <p>A set of parameters can be passed to the Entity Data Source to retrieve the corresponding instance (<code>refArgList</code> element). These parameters can be:</p> <ul style="list-style-type: none">—references to fields defined in the Entity Data Source,—attributes in the Entity Data Source (referenced using a <code>dataSourceField()</code> expression),—references to arguments passed to the Entity Data Source (referenced using an <code>arg()</code> expression)—or values (string or number) or any policy expression according to	<code>maxBearerQosProfile = BearerQosProfile(dataSourceField("MaxBQosP"));</code>



Field Type	Description	Example
	Configuration Guide for Subscription and Policies	
Entity Data Source (dataSourceField)	<p>Contains an identifier of the attribute (attrName element) in the repository where the entity can be found.</p> <p>In an LDAP repository, it must contain the name of the LDAP attribute.</p> <p>In case of the SAPC internal database, only fixed values are allowed.</p>	<pre>subscribedServices = dataSourceField("WlServ") ;</pre>
Array of Entity Data Source references	<p>Contains an array of the names of other Entity Data Sources where the information of the field is stored.</p> <p>refArgList element works as explained in former bullet.</p>	-
Concatenation	A concatenation of some of the previous elements.	See an example in Example 9 .

[Example 9](#) shows how to compose dynamic text using several input arguments:

Example 9 Concatenation Text

```
<edit-config>
  <target>
    <running />
  </target>
  <config>
    <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
      <managedElementId>1</managedElementId>
      <dnPrefix>dc=ManagedElement</dnPrefix>
      <networkManagedElementId>1</networkManagedElementId>
      <userLabel>Managed Element</userLabel>
      <PolicyControlFunction xmlns="urn:com:ericsson:ecim:sapcmom">
        <policyControlFunctionId>1</policyControlFunctionId>
        <EntityData xmlns="urn:com:ericsson:ecim:entitydatamom">
          <entityDataId>1</entityDataId>
          <EDSources xmlns="urn:com:ericsson:ecim:edsourcesmom" →
        >
          <eDSourcesId>1</eDSourcesId>
          <EDSource xmlns="urn:com:ericsson:ecim:edsourcesmom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="merge">
            <eDSourcesId>ComposeText</eDSourcesId>
            <definition>
              def ComposeText (
                arg1 = 'AccessData.subscriber.msisdn',
                arg2 = 'AccessData.subscriber.locationInfo.countryCode' )
```



```

{
  dataSource = {
    url =
      "ldap://10.20.30.222:389/ou=countries,o=company,c=es?";
    query = "(country={arg2})";
  }
  fieldDef = {
    text = "Subscriber " + arg("arg1")
    + " has started a session in country "
    + arg("arg2") + "- "
    + dataSourceField ("countryName") + ".";
  }
}
</definition>
</EDSource>
</EDSources>
</EntityData>
</PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>

```

This example, text is a field composed by chaining some constant strings, input arguments, and a dataSourceField.

For example, assuming that:

- arg1 takes the value 606777888,
- arg2 takes the value 214 and
- the following LDAP entry:

```

dn:ou=countries,o=company,c=es, country=214
countryName:Spain

```

text value results in:

"Subscriber 606777888 has started a session in country 214-Spain."

3.6 Database Access Fault Management

3.6.1 Database Access Alarms

In a scenario with external database, the SAPC can raise the following alarm:

- Connection Failure to external database.



3.6.2 Database Access Notifications

Not applicable.

3.7 Database Access Logging

In a scenario with external database, the following events are logged:

- Error fetching data from external database.
- Error storing data into external database.

3.8 Database Access Performance Management

Not applicable.



4 Reference List

Ericsson Documents

1. Configuration Guide for Subscription and Policies
2. Integration in User Data Consolidation
3. Session Context Exposure User Guide

Standards

1. Augmented BNF for Syntax Specifications: ABNF - RFC 2234
2. Lightweight Directory Access Protocol - RFC 4511