

Configuration Guide for Subscription and Policies

Ericsson Service-Aware Policy Controller

USER GUIDE

Copyright

© Ericsson España, S.A. 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	Configuration and Provisioning Overview	1
1.1	Other Conventions	2
2	Configuration Prerequisites	5
3	Subscription and Policies Data Overview	7
4	Provision Subscribers	9
4.1	Provision Subscriber Groups	9
4.1.1	Provision Global Subscriber Group	10
4.2	Provision Subscribers	11
4.2.1	Subscriber Identifiers	12
4.3	Handling of Multiple Service Offerings	14
4.4	Configure Dynamic Group Selection	15
4.5	Handling of Unknown Subscribers	17
4.5.1	Autoprovisioning	18
4.5.2	Unknown Subscriber	20
4.5.3	Unknown Subscriber EDSources pointing to the SAPC internal database	21
4.5.4	Unknown and Autoprovisioning Precedence	23
5	Provision Policies	25
5.1	Policies Basic Concepts	25
5.1.1	Rule Combining Algorithm in a Policy	27
5.1.2	Combining Algorithm in a Policy Locator	28
5.1.3	Combining Algorithm for the Subscriber Policies	29
5.2	Provisioning Policies	30
5.2.1	Subscriber Extension Data to Be Used in Policies	32
5.3	Policy Types	34
5.4	Policy Tags	45
6	Appendix A. Condition Policy Language	47
6.1	Condition Formula Examples	47
6.1.1	Reuse of Complex Expressions	47
6.2	Condition Formula Language	50
6.2.1	Condition Formula Operators	51
6.2.2	Condition Formula Expressions	53
	Reference List	63





1 Configuration and Provisioning Overview

Figure 1 shows the main parts related to configuration and provisioning in the SAPC.

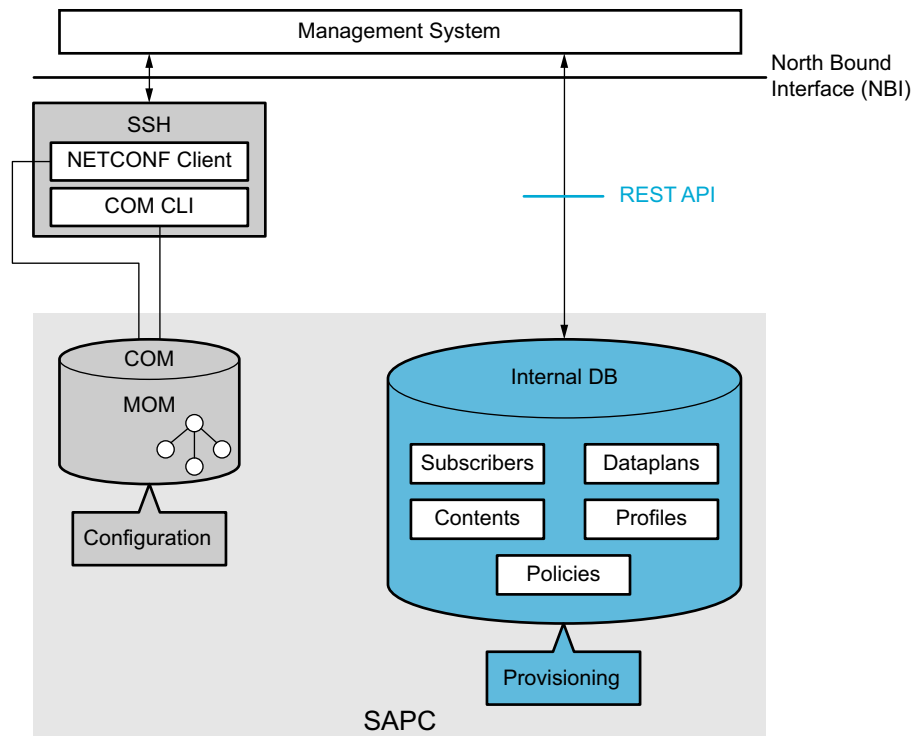


Figure 1 Configuration and Provisioning Overview

The purpose of this document is to provide a guideline for subscription and policies in the SAPC by providing some configuration and provisioning examples.

This document does not intend to be an exhaustive guide for all possibilities related to subscription and policies in the SAPC.

Furthermore, for each particular function of the SAPC (for example, Access and Charging Control, Bearer QoS Control or Fair Usage Control), specific subscription and policy data could be needed. This particular subscription and policy data can be found in the corresponding SAPC Configuration Guides document.

The complete parameter list and details of all configured options of the SAPC are included in separate documents. For more information, refer to *Managed Object Model (MOM)* and *Provisioning REST API*.

Examples in this document cover the case of data configured in the SAPC internal repository. If an external repository is used, refer to *Database Access*.



1.1 Other Conventions

This document refers to some configuration and provisioning data.

To clarify which detailed data is managed by COM or by the REST API, this document uses the following conventions:

- Configuration: whenever referring to Managed Object Class (MOC).

The detailed description of the object and attributes can be found in [Managed Object Model \(MOM\)](#).

Example: set `enableReauthsOnSubsChange` attribute in class `AppConfig`.

The tools or interfaces to manage these data in the SAPC are:

- a NETCONF interface, refer to [Ericsson NETCONF Interface](#).

The configuration examples show the NETCONF file contents, using the following syntax:

```
<edit-config>
...
<config>
  <ManagedElement xmlns="urn:com:ericsson:ecim:ComTop">
    <managedElementId>1</managedElementId>
    ...
  </ManagedElement>
</config>
</edit-config>
```

- b COM CLI, refer to [Ericsson Command-Line Interface](#).

- Provisioning: mainly subscribers, subscriber groups (dataplan), services (contents), profiles, and policy-related data. The SAPC provides a REST API for them, see [Provisioning REST API](#).

This document uses the following terminology for them: `<resource-name>` URI in the provisioning REST API.

Example: To provision subscriber groups, use the `dataplan` URI in the provisioning REST API.

Provisioning examples show HTTP operations on REST resources with the following syntax:

HTTP-Operation /resource-URI
{json content} where /resource-URI is the relative URI from the SAPC provisioning base URI detailed in [Provisioning REST API](#).

Example:



```
PUT /dataplan/Gold
{ "dataplanName" : "Gold",
  "subscribedContents" : [{"contentName" : "HTTP_Streaming",
                           "redirect" : false}]
}
```

Note: To ease provisioning operations, the SAPC provides an HTTPS CLI client named resty, refer to [Provisioning Tools](#).





2 Configuration Prerequisites

Before configuring the SAPC in an operational network, make sure that:

- CBA Components are installed
- The SAPC product software is installed
- The SAPC user performing configuration changes has thorough knowledge of the function





3 Subscription and Policies Data Overview

The main elements in the SAPC data model are Subscribers, Subscriber Groups, and Policies, which are applicable to all the SAPC functions.

Provision each subscriber or group with the desired attributes for the applicable functions that the SAPC controls, such as Access and Charging Control, Bearer QoS Control, and Fair Usage Control.

Characterizing Subscribers or Groups can be done in the following ways:

- Statically, which is called Subscriber or Subscriber Group Provisioning
- Depending on flexible conditions to be evaluated by the SAPC, with policies





4 Provision Subscribers

4.1 Provision Subscriber Groups

Subscribers sharing similar data (service offering) can be grouped. The services and policies defined for a group apply to the subscribers belonging to that group.

To provision subscribers groups, use the `dataplan` URI in the provisioning REST API.

Example 1 provisions some subscriber groups:



```
PUT /dataplan/Bronze
{
  "dataplanName" : "Bronze",
  "subscribedContents" :
  [
    {
      "contentName" : "Chat",
      "redirect" : false
    }
  ]
}

PUT /dataplan/Gold
{
  "dataplanName" : "Gold",
  "subscribedContents" :
  [
    {
      "contentName" : "Streaming",
      "redirect" : false
    },
    {
      "contentName" : "Video",
      "redirect" : false
    },
    {
      "contentName" : "Chat",
      "redirect" : false
    }
  ]
}
```

Example 1 Configuration of Subscriber Groups

Gold subscribers are enabled to execute Streaming, Video, and Chat services.

Bronze subscribers are enabled to execute only Chat services.

Gold, Bronze, and all subscribers are also enabled to execute internet service, coming from global group (see Example 2).

4.1.1 Provision Global Subscriber Group

A special subscriber group with identifier “global” and lowest group priority can be provisioned with the default values to apply for all the subscribers. All the



subscribers implicitly belong to this “global” group (without the need of setting “global” value in their attribute dataplans).

Example 2 is an example for the provisioning of the global subscriber group:

```
PUT /dataplan/global
{
  "dataplanName" : "global",
  "subscribedContents" :
  [
    {
      "contentName" : "Internet",
      "redirect" : false
    }
  ]
}
```

Example 2 Configuration of Global Subscriber Group

“global” group subscribers are enabled to execute the internet service. No services are globally blacklisted. Thus, all the subscribers provisioned in the SAPC have Internet as a subscribed service. This is why in Example 1 the groups do not contain Internet as a subscribed service.

4.2 Provision Subscribers

To provision subscribers, use the subscriber URI in the provisioning REST API.

Example 3 provisions a subscriber:

```
PUT /subscribers/34600000001
{
  "dataplan" :
  [
    {
      "dataplanName" : "Gold"
    }
  ],
  "deniedContents" : [ "Chat" ],
  "subscriberId" : "34600000001"
}
```

Example 3 Provisioning of Subscriber



Example 3 provisions subscriber “34600000001” belonging to group “Gold” but is not authorized to execute Chat service, overriding the group specifications.

Temporary Subscriptions

It is possible to apply for a subscriber certain profile data (coming from a service offering) only during a particular period. To do that, use `dataplanName`, `startDate`, and `stopDate` JSON attributes inside `dataplan`s in the subscriber URI in the provisioning REST API.

Subscriber Dates With Daylight Saving Time (DST)

For ambiguous time representations, the SAPC considers the non DST time.

An ambiguous time is a time that maps to more than one Coordinated Universal Time (UTC). It occurs when the clock time is adjusted back in time, such as during the transition from a time zone's daylight saving time to its standard time.

To avoid misleading Time of Day behaviors, do not configure times in this range.

Example 4 provisions a subscriber in Europe/Stockholm time zone with silver group which starts on the last Sunday of October 2017, 02:30 CET (non DST). This dataplan has no effect for that subscriber, on summer time, 02:30 CEST.

```
PUT /subscribers/34600000002
{
  "dataplan" :
  [
    {
      "dataplanName" : "Silver",
      "startDate" : "29-10-2017T02:30"
    }
  ],
  "deniedContents" : [ "Skype" ],
  "subscriberId" : "34600000002"
}
```

Example 4 Provision a Subscriber dataplan which applies on non DST time

4.2.1 Subscriber Identifiers

It is possible in the SAPC to use an administrative subscriber identifier to locate the subscriber profile and applicable subscriber policies different from the subscriber traffic identity received in the protocol requests.

To provision the mapping between the traffic identities and the administrative identifier, do the following steps:



1. Create an instance of EDSOURCEMOC with eDSOURCEId = SubscriberIdentity, containing exactly the content of Example 5.
2. Set administrative and traffic identifiers in subscriberId and trafficIds JSON attributes in the subscriber URI in the provisioning REST API.

Here is the exact configuration for SubscriberIdentity EDSOURCE:

```
<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:edsourcemom" xmlns:
              <eDSOURCEId>SubscriberIdentity</eDSOURCEId>
              <definition>
def SubscriberIdentity (subsId)
{
  dataSource =
  {
    url = "internaldb:";
    query = "SubscriberTrafficIdPot:{subsId}";
  }
  fieldDef =
  {
    trafficId = arg(subsId);
    adminId = dataSourceField("subscriberId");
  }
}

          </definition>
        </EDSource>
      </EDSources>
    </EntityData>
  </PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>
```

Example 5 Subscriber Identity



Warning!

If the content of SubscriberIdentity EDSrc is not exactly as shown in Example 5, the SAPC does not handle properly the mapping between traffic and administrative subscriber identifiers.

Example 6 is an example for provisioning of subscriber identities:

```
PUT /subscribers/Joe
{
  "subscriberId" : "Joe",
  "trafficIds" : [ "778373000", "joe@operator1.com" ]
}
```

Example 6 Subscriber identities mapping

Example 6 shows how to provision “Joe” Subscriber profile as the administrative identifier for this subscriber (Joe has to be used to locate his corresponding policies), with two traffic identifiers: “778373000” MSISDN and “joe@operator1.com” NAI.

When the SAPC receives several subscriber identities (several instances of Subscription-Id AVP) in the incoming Gx request, use the configurable attribute `enum subsIdType` to decide which one to use as traffic identifier.

4.3 Handling of Multiple Service Offerings

A Subscriber can be associated with several subscriptions, having each subscription different set of applicable profile data. Each subscription can have data corresponding to different functions, or to the same function, in which case it is recommended to apply a precedence or priority to resolve conflicts.

The priority of a dataplan for a subscriber can be **statically** provisioned:

- For all subscribers of a dataplan, using the `defaultPriority` attribute of the `dataplan` URI in the provisioning REST API.
- Or particularly for a subscriber using `dataplanName` and `priority` JSON attributes within `/subscribers/{subscriberId}/dataplan` URI in the provisioning REST API. Provisioning the priority for a subscriber has precedence over doing the same for any of the dataplan it may be bound to.



4.4 Configure Dynamic Group Selection

In addition to the static association of a Subscriber to a Group, it is possible to determine it **dynamically**, using Group Selection policies.

The SAPC evaluates these policies for the list of groups statically associated with the subscriber. General concepts about provisioning of policies are recommended in this point, see Section 5 on page 25.

To use Group Selection policies, follow next steps:

1. Provision at least one rule (condition) and policy associated with:

- a global policy locator: use

```
<base_URI>/locators/resources/<dataplanName>/contexts/sub  
scription
```

- b or to a subscriber locator: use

```
<base_URI>/subscribers/<subscriberId>/locators/resources/<d  
ataplanName>/contexts/subscription
```

Note: In Group Selection policies, it makes no sense to use the subscriber group locator.

Example 7 is an example of Dynamic Group Selection policy is the following:



```
PUT /rules/rDynamicGroupSelection
{
  "condition" : "((now.time < \"18:00\") && (now.time > \"20:00\"))",
  "ruleName" : "rDynamicGroupSelection"
}

PUT /policies/pDynamicGroupSelection
{
  "policyName" : "pDynamicGroupSelection",
  "ruleCombiningAlgorithm" : "permit-overrides",
  "rules" : [ "rDynamicGroupSelection" ]
}

PUT /subscribers/Joe/locators/resources/Gold/contexts/subscription
{
  "policies" : [ "pDynamicGroupSelection" ]
}

PUT /subscribers/Joe
{
  "dataplan" :
  [
    {
      "dataplanName" : "Basic"
    },
    {
      "dataplanName" : "Gold",
      "priority" : 1
    }
  ],
  "subscriberId" : "Joe"
}
```

Example 7 Configuration of Dynamic Group Selection

In Example 7, the subscriber "Joe" is statically associated with "Basic" and "Gold" groups. However, "Gold" group is only selected from 18:00 until 20:00.

Example 8 is an example of a Group Selection global policy, applicable to all Subscribers:



```

PUT /rules/rCityCells
{
  "condition" : "not( inRange(AccessData.subscriber.locationInfo.cellIdentity, \"1000,2000\")
  "ruleName" : "rCityCells"
}

PUT /policies/pCityCells
{
  "policyName" : "pCityCells",
  "ruleCombiningAlgorithm" : "permit-overrides",
  "rules" : [ "rCityCells" ]
}

PUT /locators/resources/CityGroup/contexts/subscription
{
  "policies" : [ "pCityCells" ]
}

PUT /subscribers/562500100200
{
  "dataplan" :
  [
    {
      "dataplanName" : "RusticGroup"
    },
    {
      "dataplanName" : "CitiGroup"
    }
  ],
  "subscriberId" : "562500100200"
}

```

Example 8 Configuration of Dynamic Group Selection

In Example 8, the subscriber is statically associated with “RusticGroup” and “CityGroup”. However, “CityGroup” is only selected for the subscriber in case their location does not belong to the cell identifiers (1000,2000) belonging to the rustic area.

4.5 Handling of Unknown Subscribers

The SAPC can process traffic request for subscribers that are not provisioned in internal database, in the following exclusive ways:

- 1 Autoprovisioning: The SAPC automatically provisions the subscriber when it receives the first traffic request (IP Session establishment). Afterward, the automatically provisioned subscriber works as a normal subscriber.



Note: Using this autoprovisioning function, the subscribers are only automatically created in the SAPC.

To activate autoprovisioning behavior, see Section 4.5.1 on page 18.

- 2 Or, use a shared profile, see Section 4.5.2 on page 20

4.5.1 Autoprovisioning

To use autoprovisioning, do the following:

- 1 If the SAPC receives several subscriber identities (several instances of Subscription-Id AVP) in the incoming Gx request, to decide which subscriber identity is used for the autoprovisioned subscriber use the attribute `enum subsIdType`.
- 2
 - a If needed to select the subscriber group associated with the subscriber, depending on dynamic conditions, use autoprovisioning policies, see Section 4.5.1.1 on page 18.
 - b Or, if not needed dynamic conditions (or for the cases when the condition evaluates to false), create “auto” subscriber group.
- 3 If it is needed to differentiate (personalize) the subscriber data of the autoprovisioned subscriber once it is autoprovisioned, provision data in the corresponding subscriber (after the first CCR message for the subscriber automatically creates it).

4.5.1.1 Provision Autoprovisioning Policies

Some previous general concepts about provisioning of policies are recommended in this point, see Section 5 on page 25.

To configure autoprovisioning scenario by policies, do the following:

- 1 Add one or more groups to be used as groups for automatically created subscribers.
- 2 Add **Autoprovision** policies at global policy locator.

Example 9 shows the configuration using an autoprovisioning policy.

```
PUT /rules/rAutoprovisioning
{
  "condition" : "(AccessData.bearer.accessType == 1004)",
  "outputAttributes" :
  [
    {
```



```

        "attrName" : "dataplan",
        "attrValue" : "\"4g-group\"",
        "result" : "permit"
    }
],
"ruleName" : "rAutoprovisioning"
}

PUT /policies/pAutoprovisioning
{
    "policyName" : "pAutoprovisioning",
    "ruleCombiningAlgorithm" : "permit-overrides",
    "rules" : [ "rAutoprovisioning" ]
}

PUT /locators/resources/any/contexts/autoprovisioning
{
    "policies" : [ "pAutoprovisioning" ]
}

PUT /dataplan/4g-group
{
    "dataplanName" : "4g-group"
}

PUT /dataplan/AutoProvFixed
{
    "dataplanName" : "AutoProvFixed",
    "usageLimits" :
    [
        {
            "absoluteLimits" :
            {
                "dlVolume" : 1048576,
                "resetPeriod" :
                {
                    "volume" : "monthly"
                }
            },
            "description" : "Total traffic"
        }
    ]
}

```

Example 9 Autoprovisioning policies

In Example 9, the SAPC associates the group “4g-group” to autoprovisioned subscribers when the access type is E-UTRAN (rule condition (AccessData.bearer.accessType == 1004)). It also provisions usage limits for the “4g-group” dataplan.



4.5.1.2 Subscriber Group for Autoprovisioning

If there is no need of a dynamic condition to select the group for the autoprovisioned subscribers, it is possible to assign statically the special auto subscriber group.

To do so, create a dataplan called “auto”.

Example 10 shows how for autoprovisioned subscribers following data apply: 1 GB data volume per month for total traffic.

```
PUT /dataplan/auto
{
  "dataplanName" : "auto",
  "usageLimits" :
  [
    {
      "absoluteLimits" :
      {
        "dlVolume" : 1048576,
        "resetPeriod" :
        {
          "volume" : "monthly"
        }
      },
      "description" : "Total traffic"
    }
  ]
}
```

Example 10 Autoprovisioning Configuration

4.5.1.3 Autoprovisioning and Dynamic Group Selection

The SAPC can assign, statically or dynamically, autoprovisioned subscribers to a group. In both cases, the subscriber only belongs to one group, and therefore, the dynamic Group Selection (Section 4.4 on page 14) does not apply.

4.5.2 Unknown Subscriber

For scenarios where it is not needed subscriber differentiation, it is possible to share the same profile (called “unknown”): the SAPC can process requests for a subscriber identifier that it is not found.

If the “unknown” Subscriber is not provisioned, and the SAPC receives a request containing a subscriber identifier not found, the SAPC rejects the request specifying that the subscriber identity is unknown.

To share the common profile for unknown subscribers, do the following:



- 1 Provision a subscriber called “unknown” (administrative identifier), and the desired services and data to apply for all the unknown subscribers.
- 2 Configure the subscriber unknown related Entity Data Sources: create in COM an instance of the following EDSources:
 - eDataSourceId = SubscriberUnknown
 - eDataSourceId = GroupsToSubscriberUnknown

Note: If some data are not needed for the “unknown” subscriber, it is possible to simplify previous EDSources, see **Not Needed Data in Database Access**.

Example:

```
def SubscriberUnknown ( argId )
{
  dataSource =
  {
    url = "internaldb:";
    query = ...;
  }
  fieldDef =
  {
    ...
    usageLimits = "";
    sms = "";
    sharedDataplan = dataSourceField("sharedDataplanId");
  }
}
```

The example above shows the configuration for SubscriberUnknown EDSources, where it does not use Fair Usage, neither end-user notifications.

4.5.3 Unknown Subscriber EDSources pointing to the SAPC internal database

And next, the exact configuration for SubscriberUnknown EDSources pointing to the SAPC internal database:

```
<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:edsourcemom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <eDSourceId>SubscriberUnknown</eDSourceId>
    <definition>
def SubscriberUnknown ( argId )
{
  dataSource =
  {
    url = "internaldb:";
    query = "SubscriberPot:{argId}";
  }
  fieldDef =
  {
    id = dataSourceField("id");
    groups = dataSourceField("dataplan");
    trafficIds = dataSourceField("trafficIds");
    sharedDataplan = dataSourceField("sharedDataplanId");
    subscribedServices = dataSourceField("subscribedContents");
    blacklistServices = dataSourceField("deniedContents");
    contentFilteringProfileId = dataSourceField("contentFiltering");
    chargingProfile = SubsChargingProfile(dataSourceField("subscriberChargingProfile"));
    chargingSystem = OnlineChargingSystemProfile(dataSourceField("onlineChargingSystem"));
    customerId = dataSourceField("customerId");
    maxBearerQosProfile = BearerQosProfile(dataSourceField("maxBearerQosProfile"));
    minBearerQosProfile = BearerQosProfile(dataSourceField("minBearerQosProfile"));
    servicesToRedirect = dataSourceField("redirectContents");
    presenceReportingAreaNames = dataSourceField("presenceReportingAreaNames");
    usageLimits = dataSourceField("usageLimits");
    sms = dataSourceField("smsDestinations");
    eventTriggers = dataSourceField("eventTriggers");
    pdnGwListName = PdnGwListProfile(dataSourceField("pdnGwListName"));
    spid = dataSourceField("spid");
  }
}
    </definition>
    </EDSource>
    <EDSource xmlns="urn:com:ericsson:ecim:edsourcemom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <eDSourceId>GroupsToSubscriberUnknown</eDSourceId>
    <definition>
def GroupsToSubscriberUnknown( subsId, groupId )
{
  dataSource =
  {
    url = "internaldb:";
    query = "SubscriberPot:{subsId}";
  }
  fieldDef =

```



```

    {
      priority = dataSourceField("dataplan:{groupId}:prio");
      startDate = dataSourceField("dataplan:{groupId}:start_date");
      endDate = dataSourceField("dataplan:{groupId}:end_date");
    }
  </definition>
</EDSource>
</EDSources>
</EntityData>
</PolicyControlFunction>
</ManagedElement>
</config>
</edit-config>

```

Example 11 Subscriber Unknown EDSources

4.5.4

Unknown and Autoprovisioning Precedence

The use of “unknown” subscriber and autoprovisioned subscribers (static assignation of autoprovisioned subscriber group) are mutually exclusive. If the provisioning includes “auto” subscriber group, unknown subscriber does not apply.

To autoprovision some subscribers but apply the unknown for some others, use autoprovisioning policies: when the condition of such autoprovisioning policies is not fulfilled, but the subscriber with identifier “unknown” is provisioned, the SAPC uses this default profile for not provisioned subscribers.





5 Provision Policies

5.1 Policies Basic Concepts

The following video shows the basic concepts of policies.



©Ericsson España 2016. All rights reserved. No part of this material may be reproduced in any form without the written permission of the copyright owner. The contents of this material 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 material.
The procedures demonstrated in this video may not reflect a real-time environment.

Figure 2 Policies Basic Concepts Video

The main policies data are:

- Policy: a set of rules



- Rule: contains a boolean expression that evaluates information (policy tags) and gives as result a true or a false.
- Policy Locator: contains a set of policies. It is identified by the following elements:
 - Context: represents the policy category or policy type. It indicates for which part of business logic, the SAPC uses that policy; for which function is the policy evaluated (for example QoS, authorize services, autoprovisioning).
 - Resource: indicates for what the policy applies.
 - Subject: indicates for whom the policy applies.

Next video shows how the SAPC evaluates rules and policies.



POLICIES EVALUATION

› How does SAPC **evaluate** Policies?



©Ericsson España 2016. All rights reserved. No part of this material may be reproduced in any form without the written permission of the copyright owner.

The contents of this material 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 material.

The procedures demonstrated in this video may not reflect a real-time environment.

Figure 3 Policies Evaluation

5.1.1 Rule Combining Algorithm in a Policy

Indicates how to select the result of the rules associated to one policy.

The output attributes returned are the ones associated to the rule that determined the result.

Algorithms for **decision**:

- Single result: when only one result of all the rules within the policy is returned, and there are results contradictory for a resource:
 - Permit overrides: the result is “permit”, if any rule evaluates to permit.



- Deny overrides: the result is “deny” if any rule evaluates to deny.
- Multiple result: when it is wanted that all the rules of a policy are evaluated, use all permit algorithm. In that case, the output attributes of all the rules that evaluates to permit are returned.

Algorithms for **classification**: a policy is considered fulfilled:

- If all the evaluated items match with any of the rules of the policy
- And if all its rules match with any of the evaluated items

And the SAPC provides the following options:

- single match: each rule in the policy, once matched with one of the evaluated items, is not used in the evaluation of the remaining items.
- multiple match: the rule in the policy can be used in the evaluation of each of the items, even if it already matched with one of them.

5.1.2

Combining Algorithm in a Policy Locator

The algorithm to solve conflicts among the list of policies inside a policy locator is fixed (not configured):

- Permit overrides for single result evaluation policies
- All permit for multiple result evaluation policies

The next video shows how the SAPC selects the policies applicable to a subscriber.

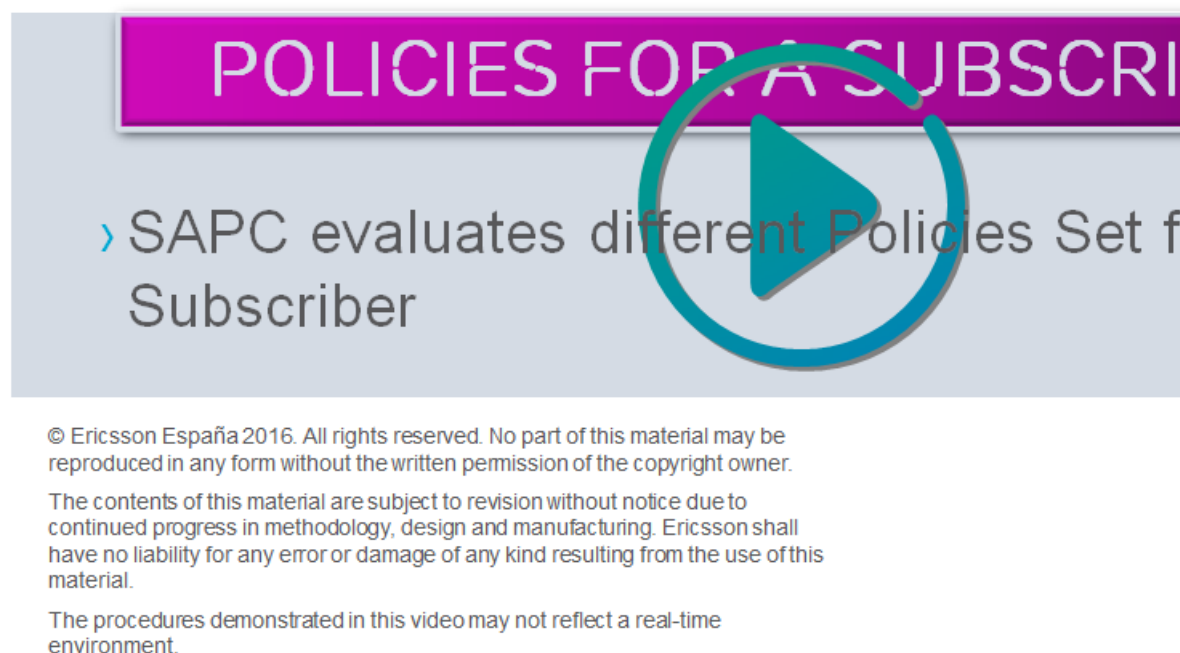


Figure 4 Selection of Data for a Subscriber

5.1.3 Combining Algorithm for the Subscriber Policies

For a Subscriber the SAPC evaluates in order the set of policies belonging to several Policy Locators (different subjects).

The algorithm to solve conflicts among the different locator policies is fixed (not configured):

- Permit overrides for single result evaluation policies.
- All permit for multiple result evaluation policies.

The next video explains how policies and time of day conditions work.



© Ericsson España 2015. All rights reserved. No part of this material may be reproduced in any form without the written permission of the copyright owner.

The contents of this material 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 material.

The procedures demonstrated in this video may not reflect a real-time environment.

Figure 5 Policies and Time of Day

5.2 Provisioning Policies

To provision a policy, create at least:

- A policy locator
 - **global policy locator:** use it for policies applicable globally for all subscribers (independently of subscriber or group), use the following REST URI:



```
<base_URI>/locators/resources/<resourceName>/contexts/<context-name>
```

Example: Service QoS policy locator for whatsapp service:

```
<base_URI>/locators/resources/whatsapp/contexts/qos
```

- **subscriber group locator:** for conditions applicable to a **Subscriber Group** (and therefore to subscribers belonging to such group), use the following REST URI:

```
<base_URI>/dataplan/<dataplanName>/locators/resources/<resourceName>/contexts/<contextName>
```

Example: Service QoS policy locator for whatsapp service and gold subscriber group:

```
<base_URI>/dataplan/gold/locators/resources/whatsapp/contexts/qos
```

Warning!

Ericsson discourages the use of policies at subscriber group level, when using as well Dynamic Group Selection function. When the group is not active because of Dynamic Group Selection, other policies configured for the group, are not evaluated. This may lead to unexpected behavior (for example a service not authorized, a different QoS than the desired, or the same end-user notification sent twice or not sent at all).

- **subscriber locator:** for conditions applicable to a **Subscriber**, use the following REST URI:

```
<base_URI>/subscribers/<subscriberId>/locators/resources/<resourceName>/contexts/<contextName>
```

Example: Service QoS policy locator for whatsapp service and subscriber john:

```
<base_URI>/subscribers/john/locators/resources/whatsapp/contexts/qos
```

Ericsson only recommends this option when there is a real need of individual personalization at Subscriber level, as it complicates the provisioning of such policies.

- A policy associated with the policy locator: using policy URI in the provisioning REST API.
- A rule associated with the policy: use rule URI in the provisioning REST API.



Set the condition inside the rule (using the language in Section 6 on page 47), and depending on the policy type, the outputAttributes.

Caution!

The SAPC does not consider immediately changes done in these parameters, but refreshes their values every 5 minutes.

5.2.1

Subscriber Extension Data to Be Used in Policies

Operators can want to use their own extra data (not known in advance by the SAPC) to be evaluated in the SAPC policy conditions. So, from the SAPC point of view, it is possible to extend Subscriber entry with extra attributes to the ones considered in Table 13.

Follow next steps for each additional attribute in the subscriber profile (example attribute: "age"):

- 1 Reconfigure Subscriber EDSOURCE to add as a new fieldDef the new attribute.

Note: Do not remove any of the rest of fieldDefs set at installation time.
- 2 Provision the name and value for the new attribute in operatorSpecificInfos of the subscriber REST URI.

Example 12 is an example to reconfigure Subscriber EDSOURCE:



```

<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:edsourcemom"
              xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="merge">
              <eDSourcesId>Subscriber</eDSourcesId>
            </EDSource>
          </EDSources>
        </EntityData>
      </PolicyControlFunction>
    </ManagedElement>
  </config>
</edit-config>

def Subscriber( argId ) {
  dataSource = {
    url = "internaldb:";
    query = "SubscriberPot:{argId}";
  }
  fieldDef = {
    ...
    age = dataSourceField("name:age");
  }
}

```

Example 12 Subscriber Entity Data Source

Example 13 shows the provision of the attribute age for the subscriber 3460000011 with the value 20.



```
PUT /subscribers/34600000011
{
  "operatorSpecificInfos" :
  [
    {
      "attributeName" : "age",
      "attributeValue" : "20"
    }
  ],
  "subscriberId" : "34600000011"
}
```

Example 13 Operator Specific Info Provisioning

Subscriber.age is ready to be used within policy conditions.

5.3 Policy Types

This section includes as summary, the complete list of the SAPC policy types that can be used for the various functions that the SAPC provides.

The particular use and details of each policy type is covered in the corresponding SAPC Configuration Guides document.

Table 1 Subscription Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Auto provisioning	autoprovisioning	any	-	permit dataplan "<dataplanId>"	Only has sense to use Global Policy Locator
Dynamic Group Selection	subscription	<dataplanId>	<subscriberId>	-	No sense to use Policy Locator using as Subject = the Dataplan
Charging System	charging-system	any	-	permit charging-system "<chargingSystemName>"	Only has sense to use Global Policy Locator



Table 2 Accumulation Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Conditional Accumulation	accumulation	reporting-group <reporting-group-id>	<subscriberId> <dataplanId>	-	Type I = Only policies, no qualification Used to accumulate or not depending on flexible policy conditions
Dynamic Group Selection	subscription	reporting-group.<counterId> <reporting-group-id>.<counter-Id>	<subscriberId> <dataplanId>	-	Type I = Only policies, no qualification Used to accumulate on different counters.

Table 3 Access Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Access Control (Service Authorization) Auto provisioning	autoprovisioning	any	-	permit dataplan "<dataplanId>"	Only has sense to use Global Policy Locator



Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Dynamic Group Selection	subscription	<dataplanId>	<subscriberId>	-	No sense to use Policy Locator using as Subject = the Dataplan
Charging System	charging-system	any	-	permit charging-system "<chargingSystemName>"	Only has sense to use Global Policy Locator

Table 4 Access Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Access Control (Service Authorization) Access	access	<contentId>	<subscriberId> <dataplanId>		Type I = Only policies, qualification Gx Conditions: Subscriber Data Access Data ToD
Access Control (Static service qualification) Static Access	static-access	<contentId>	<subscriberId> <dataplanId>	permit pcc-rule-id "<pccRuleName>"	Type I = Only policies, qualification Gx Conditions: Subscriber Data Access Data ToD



Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Rule Space selection Service Domain	access	service-domain	<subscriberId> <dataplanId>	permit rule-space "<ruleSpaceName>"	Ericsson Gx+
IP-CAN Session Access	access	ip-can-session	<subscriberId> <dataplanId>	-	Type I = Only policies, qualification Gx Conditions: Subscriber Data Access Data ToD

Table 5 Charging and Content Filtering Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Service Charging	charging	<contentId>	<subscriberId> <dataplanId>	permit charging ServiceChargingProfile ["<chargingProfileName>"]	Type II = Mixing policies and qualification Gx Conditions: Access Data Subscriber ToD



Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Subscriber Charging	charging	any	<subscriber-id> <dataplan-id>	permit subs-charging ServiceChargingProfile ["<chargingProfileName>"]	Type II = Mixing policies and qualification Gx Conditions: Access Data Subscriber Time conditions (but no ToD reauthorization)
Content Filtering	content-filtering	service-domain	<subscriberId> <dataplanId>	permit content-filtering-id "<contentFilteringValues>"	Type II = Mixing policies and qualification Used to return Content-Filtering in Ericsson Gx+



Table 6 QoS Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Bearer QoS Control QoS for Service	qos	<contentId>	<subscriberId> <dataplanId>	permit qos ServiceQosProfile ["<qosProfileName>"]	Type II = Mixing policies and qualification Gx Conditions: Subscriber Data Access Data ToD
Bearer QoS Control QoS for Service	qos	ip-can-session	<subscriberId> <dataplanId>	permit max-qos BearerQosProfile ["<qosProfileName>"] or qos_prof_expression permit min-qos BearerQosProfile ["<qosProfileName>"] or qos_prof_expression	Type II = Mixing policies and qualification Gx Conditions: Subscriber Data Access Data ToD



Table 7 Dynamic Policy Control (Rx) Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Dynamic Service Classification	service-classification	application	-	service <contentId>	Only policies, no qualification Gx - Rx binding Algorithms: single match, multiple match Conditions: - AfData
Access Control (Dynamic Service Authorization) Access	access	<contentId>	<subscriberId> <dataplanId>	-	Mixing policies and qualification Conditions: Access Data Subscriber ToD
Bearer QoS Control (Dynamic Service Qualification) QoS for Service	qos	<contentId>	<subscriberId> <dataplanId>	permit qos ServiceQosProfile ["<qosProfileName>"]	Mixing policies and qualification Conditions: media components (AfData) Access Data Subscriber ToD



Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
(Dynamic Service Qualification) Service Charging	charging	<contentId>	<subscriberId> <dataplanId>	permit charging ServiceChargingProfile ["<chargingProfileName>"]	Mixing policies and qualification Conditions: media components (AfData) Access Data Subscriber ToD
Bearer QoS Control QoS for Service	qos	ip-can-session	<subscriberId> <dataplanId>	permit max-qos BearerQosProfile ["<qosProfileName>"] or qos_prof_expression permit min-qos BearerQosProfile ["<qosProfileName>"] or qos_prof_expression	Mixing policies and qualification Conditions: media components (AfData) Access Data Subscriber ToD



Table 8 Mobility Based Policy Control (Smp) Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Smp Session control	access	sx-session	<subscriberId> <dataplanId>	-	Type I = Only policies, no qualification Smp Conditions: Access Data Subscriber
PDN GW Selection	pdn-gw	any	<subscriberId> <dataplanId>	permit pdn-gw-list "<pdnGwList name value>"	Type II = Mixing policies and qualification Smp Conditions: Access Data Subscriber
SPID Selection	spid	any	<subscriberId> <dataplanId>	permit spid value	Type II = Mixing policies and qualification Smp Conditions: Access Data Subscriber



Table 9 Notification Related Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Notifications	notification	any	<subscriberId> <dataplanId>	permit notification "<Notification message>" permit notification NotifReceiver "<Notification message>"	Type III = All Permit _Used to send Subscriber notifications Algorithms: all permit

Table 10 Presence Reporting Area in Gx Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Notifications	notification	any	<subscriberId> <dataplanId>	permit notification "<Notification message>" permit notification NotifReceiver "<Notification message>"	Type III = All Permit _Used to send Subscriber notifications Algorithms: all permit



Table 11 Presence Reporting Area in Gx Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Presence Reporting Area Selection	location	any	<subscriberId> <dataplanId>	permit presence-area PraProfile ["<presenceAreaName>"]	Type II = Mixing policies and qualification Gx Conditions: Access Data Subscriber Time conditions (but no ToD reauthorization)

Table 12 Event Triggers Selection Policies

Policy Type	Policy Locator			Output Attributes	Comments
	Context	Resource	Subject		
Event Triggers Selection	event-triggers	any	<subscriberId> <dataplanId>	permit event-triggers "<list of event trigger values>"	Type IV= Mixing AllPermit policies and qualification Gx Conditions: Access Data Subscriber Media component (AfData) ToD Algorithms:all permit List of event triggers in CSV format



5.4 Policy Tags

The conditions that the SAPC evaluates in policies are set in condition attribute of a rule. It is possible to use the following data (policy tags):

- Subscriber or subscriber group data
- Operators and functions, defined in Section 6 on page 47.
- Data from the access received through protocol messages (or from internal the SAPC session state), that depend on the particular scenario in which the SAPC is deployed.

Note: Depending on the SAPC function, there are different applicable policy tags, which can be found in each SAPC Configuration Guides document.

Subscription Related Policy Tags

Tags related to the subscriber profile can be used in the condition formula of rules. The data makes reference to data provisioned for the subscriber to which the rule to be evaluated applies.

Table 13 Subscription Related Tags

Tag	Return Type	Possible Values	Comments
Subscriber.x	Any	Any	x is any of the fieldDefs defined in Subscriber EDSources (see Section 5.2.1 on page 32)
Subscriber.groups	String		List of group identifiers (dataplan) statically provisioned in the Subscriber profile.
Subscription.group["groupname"].isActive (1)	Boolean	true false	Indicates if the group (dataplan) is active for the subscriber, as result of subscription information (Subscription.group["groupname"].isSubscribed equal to true) and Dynamic Group Selection policies (see Section 4.4 on page 14).
Subscription.group["groupname"].isSubscribed	Boolean	true false	Indicates if the group (dataplan) is statically associated with the subscriber and current date is between start and end subscription date.

(1) Because Subscription.group["groupname"].isActive itself depends on the result of Dynamic Group Selection, avoid using it as a condition of Dynamic Group Selection policies. That could cause uncertain results.





6 Appendix A. Condition Policy Language

6.1 Condition Formula Examples

This chapter contains examples of possible expressions that can be used within a condition formula. They are just examples, and their correctness (syntactically and semantically) depends on for which functionality is used the condition.

- The condition formula evaluation result is true if the subscriber identifier is equal to 12345678 and has subscribed “service1” or “service2”.

```
(Subscriber.id == 12345678) &&
  (Subscriber.subscribedServices[0] == "service1"
   || Subscriber.subscribedServices[1] == "service2" )
```
- The condition formula evaluation result is true if the current day is equal to 3 and the subscriber IP Address received in the service control request message is equal to 10.21.164.146.

```
( now.day == 3 ) &&
  (AccessData.subscriber.ueIpAddress == "10.21.164.146" )
```
- The condition formula evaluation result is true if the string returned by the function is equal to “subs”.

```
StrDRight(Subscriber.id, AccessData.subscriber.id)
== "subs"
```

6.1.1 Reuse of Complex Expressions

A condition formula can be composed of several logical expressions. If needed to reuse the same complex expression in different condition formulas, it is possible to use Entity Data Source mechanism. This makes easier to read and maintain future changes in the complex expression. For details, refer to [Database Access](#).

To do so, perform the following configuration steps:

1. Create an auxiliary Entity Data Source (whose dataSource points to internal database).
2. Include the logical expression to be reused as default argument of the Entity Data Source.
3. Assign the argument to a fieldDef of the Entity Data Source.
4. Include the fieldDef in the condition formulas that want to reuse the logical expression.

Note: Ericsson discourages the reuse of expressions containing boolean time conditions.

For example, an operator wants to define different operation modes called **Full**, **Restricted** and **Default** and that only one of them is applied (for example to authorize a service, or any other policy control managed by the SAPC).

- **Restricted** mode condition is that the subscriber is located in a particular area in local operator network (shown in Example 14), and it is not in **Full** mode.
- The characteristics defining **Full** mode are: the subscriber is using the local operator network and the subscriber terminal equipment is a compatible smart-phone. Using policy tag this is a complex expression, such as the one shown in Example 15:
- And **Default** mode means not being **Full** neither **Restricted**.

Example 14 is an example to reuse of the **Restricted** mode expression:

```
<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:edsourcemom" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
                            <eDSourceId>RestrictedMode</eDSourceId>
                            <definition>
                                def RestrictedMode
                                (
                                    expression = '(
                                        inRange(AccessData.subscriber.locationInfo.sgsnAddress,
                                            "10.16.120.0/32,10.16.130.0/32")
                                    )',
                                )
                                {
                                    dataSource =
                                    {
                                        url = "internaldb";
                                        query = "";
                                    }
                                    fieldDef =
                                    {
                                        isActive = dataSourceField("expression");
                                    }
                                }
                            </definition>
                        </EDSource>
                    </EDSources>
                </EntityData>
            </PolicyControlFunction>
        </ManagedElement>
    </config>
</edit-config>
```

Example 14 Definition of an Entity Data Source for reusable expression.

Example 15 shows how to define the reuse of **Full**:

Example 16 is the example of different condition formulas for full, restricted and default modes, showing the reuse:



```

PUT /rules/defaultMode
{
  "condition" : "not(FullMode.isActive)&& not(RestrictedMode.isActive)",
  "ruleName" : "defaultMode"
}

PUT /rules/fullMode
{
  "condition" : "FullMode.isActive",
  "ruleName" : "fullMode"
}

PUT /rules/restrictedMode
{
  "condition" : "not(FullMode.isActive)&& RestrictedMode.isActive",
  "ruleName" : "restrictedMode"
}

```

Example 16 Condition formulas reusing expressions defined in Entity Data Source.

6.2 Condition Formula Language

A condition formula is composed of expressions related to operators whose evaluation result is a boolean value.

The language of the condition formula according to Backus-Naur Form (BNF) specification (refer to Backus Naur Form - RFC 2234) has the following format:

```

<ConditionFormulaExpression> ::=
    <Expression> binOp <Expression> |
    <Expression>

<Expression> ::=
    keyword <AccessToStructure> |
    keyword "(" <FunctionParams> ")" |
    "(" <Expression> ")" |
    resource-id

<AccessToStructure> ::=
    "." keyword <AccessToStructure> |
    "[" <Expression> "]" <AccessToStructure>

<FunctionParams> ::=
    <Expression> |
    <Expression> "," <FunctionParams>

binOp: "+" | "-" | "*" | "==" | "!=" | "&" | "|" | "&&" |
      "||" | "<" | ">"
resource-id: double quoted string | positive integer
keyword: string

```

The meaning of the condition formula language elements is the following:



— Expression represents an operand of the condition formula. An expression is composed of:

- keyword element represents the allowed fields, tags, or functions.
- `AccessToStructure` represents an expression constructed with tags. Tags can refer to final elements or structured elements whose fields are referenced using the dot and then the name of the field. Tags can require parameters to be passed to them, each of which is passed enclosed between []. (Example: `MobileTerminalDatabase["K700"].isWap`. In this example, "K700" is a parameter passed to the `MobileTerminalDatabase` tag and `isWap` is a field of the tag).

When fields are multivalued (two or more values), the `<Expression>` between square brackets is used to select one of the possible values.

(Example: `Subscriber.subscribedServices[1]=="premium"`. In the example, the tag `Subscribers` has a field that contains two values: "basic" and "premium").

When no square brackets are used in the multivalued field, the SAPC evaluates the condition formula as a unique string containing all the values separated by commas.

(Example: `Subscriber.subscribedServices=="basic,premium"`).

- `FunctionParams` represents the arguments of a function.
- `resource-id` represents fixed string or integer values.

Note: Integers values have at least 64 bits. Its range is from -9223372036854775807 to 9223372036854775807.

Regarding string values the SAPC is case-sensitive. It is not possible to configure strings including double quote (") character.

For further description of expression elements, see Section 6.2.2 on page 52.

— `binOp` represents the operators used to indicate the operation among different expressions. See Section 6.2.1 on page 51.

The parenthesis are used to perform the nesting between different expressions and to control the precedence between these expressions. The precedence in the evaluation is from **left to right**.

6.2.1 Condition Formula Operators

The formula operators are used to combine different expressions within the condition that the SAPC evaluates.



Warning!

To avoid unexpected evaluation results, Ericsson recommends to explicitly use parenthesis.

If parenthesis are not explicitly used, there is no precedence in the operators, and the SAPC evaluates the condition from left to right.

The following example shows how the SAPC evaluates operators if no parenthesis are used:

```
Subscriber.id == "12345678" && AccessData.bearer.accessType == 1001
```

The previous condition formula is evaluated as:

```
Subscriber.id == ("12345678" && (AccessData.bearer.accessType == 1001))
```

Table 14 shows the operators for the condition formula :

Table 14 Condition Formula Operators

Operators	Operation Meaning
+	Adds the integer values of 2 expressions. ⁽¹⁾
-	Subtracts the integer values of 2 expressions.
*	Multiplies the integer values of 2 expressions. ⁽²⁾
==	Checks the equality of its operands.
!=	Checks the inequality of its operands.
&&	Logical AND operation between 2 expressions.
	Logical OR operation between 2 expressions.
&	Bitwise AND operation between 2 expressions.
	Bitwise OR operation between 2 expressions.
<	Expression lower than other expression.
>	Expression higher than other expression.

(1) Logical addition of 2 logical expressions is also supported: op1 + op2 returns op1 when op2 is false.

(2) Logical multiplication of 2 logical expressions is also supported: op1*op2 returns op1 when op2 is true.



6.2.2 Condition Formula Expressions

According to the condition formula language specification, an Expression can be composed of a set of tags, functions, and fixed values.

6.2.2.1 Tags

A tag is a name representing a concept and is used to construct an expression of the condition formula.

There are tags that can be used for taking decisions related to any the SAPC function (for example time and date). Other tags are only allowed depending on the SAPC node functionality (for such cases, refer to the appropriate Configuration Guide document).

6.2.2.2 ToD Tags

Time and date tags can be used in the SAPC to take decisions based on flexible time of date conditions.

Table 15 Time and Date Tags

Tag	Return Type	Possible Return Values	Comments
<code>now.offset</code>	Integer	any	Number of seconds elapsed since 00:00:00 January 1, 1970.
<code>now.time</code>	String	any	Time of the day, expressed in following way: “hh:mm[:ss]” Where hh = 2 digits for hour, 00 - 23 (am/pm NOT allowed) mm = 2 digits for minutes, 00-59 ss = 2 digits for seconds, 00-59. Optional, when not included, default seconds are 00.
<code>now.day</code>	Integer	1–31	Day of month



Tag	Return Type	Possible Return Values	Comments
<code>now.dayOfWeek</code>	Integer	1–7	Day of week: 1: Monday 2: Tuesday 3: Wednesday 4: Thursday 5: Friday 6: Saturday 7: Sunday
<code>now.month</code>	Integer	1–12	Month
<code>now.year</code>	Integer	any (all digits of the year must be specified)	Year

These tags refer to the SAPC local date and time. In mobile scenarios, if the SAPC receives the 3GPP-MS-TimeZone AVP and the SAPC is configured to consider it (`enableMsTimeZone = true`), its value is applied to the tags.

When the SAPC evaluates conditions including these tags, it also calculates an internal validity (number of seconds) for which the boolean result condition keeps its boolean result (considering the applied combining algorithm, see Section 5.1.1 on page 27).

Warning!

"00:00:00" is the first second of a day, so, `now.time < "00:00"` results always in false. To refer to midnight (that is, last instant of a day), use "23:59:59".

Some examples for using time and date tags in the condition formula are provided next:

- Next condition formula evaluates to true if the month is November:

```
now.month == 11
```

- Next condition formula evaluates to true from 17:30:01

```
(now.time > "17:30")
```



- Next condition formula evaluates to true from 17:30:01, until 18:29:59
`(now.time > "17:30") && (now.time < "18:30")`
- Next condition formula evaluates to true from 10:00:01 to 11:59:59 and from 18:00:01 to 19:59:59
`((now.time > "10:00") && (now.time < "12:00")) ||
 ((now.time > "18:00") && (now.time < "20:00"))`
- Next condition formula evaluates to true from 20:00:01 to 07:59:59 of the following day
`((now.time < "08:00") || (now.time > "20:00"))`
- Next condition formula evaluates to true in year 2013:
`now.year == 2013`
- Next condition formula evaluates to true the first day of every month
`now.day == 1`
- Next condition formula evaluates to true on week-ends (saturday and sunday)
`(now.dayOfWeek == 6) || (now.dayOfWeek == 7)`

6.2.2.3

Functions

The arguments of the functions can be an allowed combination of tags, a fixed value or an invocation to other function, considering that the type of these data is equal to the argument type defined for the function.

Warning!

When using a list of elements, do not use "," within any element of the list because the elements are separated by ",". Otherwise, the policy evaluation leads to an unexpected result.

Table 16 shows the supported functions:



Table 16 Condition Functions

Function	Return Type	Arguments	Comments
<code>contains(array list, string elem)</code>	Boolean	<code>list</code> : a list of elements (elements are separated by “,”). <code>elem</code> : the element to find inside the list.	Returns true if the passed element is included in the past list. False otherwise.
<code>firstCharsOf(string st, integer n)</code>	String	<code>st</code> : the input string. <code>n</code> : the number of characters to be selected from the input string.	Returns the first n characters starting to count from the left of the string passed as the first argument.
<code>inPeriod(integer time, string periods)</code>	Boolean	<code>time</code> : <code>now.offset</code> See Table 15 <code>periods</code> : list of date and time periods.	Returns true if at least one period contains the passed time. False otherwise. This function also returns a validity (Section 6.2.2.2 on page 53) with the number of seconds for the next change. For the format of the periods definition, see Section 6.2.2.3.1 on page 58.



Function	Return Type	Arguments	Comments
<code>inRange(string elem, array ranges)</code>	Boolean	elem: the input element. ⁽¹⁾ ranges: a list of ranges.	<p>Returns true if the passed “elem” is included in any of the specified “range”s. In case the “elem” is a range, it returns true if the range is included in any of the specified “range”s. In case the “elem” is a list or a list of ranges, the function returns true if all the list elements are included in any of the specified “range”s.</p> <p>False otherwise.</p> <p>To specify ranges use “-” separator, for example: 1-5. To specify lists use “,” separator, for example: 1,5 or 1-5, 7-10.</p> <p>IPv4 addresses in dot format (either using mask or range notation) and IPv6 prefixes in colon notation (preferred format) are also accepted. For example: 192.168.0.0/24 192.168.0.1-192.168.0.254</p>
<code>lastCharsOf(string st, integer n)</code>	String	st: the input string. n: the number of characters to be selected from the input string.	Returns the last n characters starting to count from the right of the string passed as the first argument.
<code>not(boolean bool)</code>	Boolean	bool: the condition to negate.	Applies the not boolean operation on the boolean condition passed.
<code>strcat(string str1, string str2)</code>	String	str1, str2 input strings	Returns the string resulting of joining the strings passed as input.



Function	Return Type	Arguments	Comments
<code>substr(string str, int index, int length)</code>	String	<code>str</code> : input string <code>index</code> : index position, starting from zero <code>length</code> : number of characters to be selected from the input string	Returns the substring extracted from the input string, from the specified index position (included) until the number of characters indicated by <code>length</code> , or until the end of the input string if <code>length</code> ≤ 0 .
<code>StrDLeft(string st, string delimiter)</code>	String	<code>st</code> : the input string. <code>delimiter</code> : the delimiter to find inside the input string.	Tries to find the delimiter passed in the second argument inside the string in the first argument. If found, it returns the string composed of the characters to the left of the delimiter in the input string.
<code>StrDRight(string st, string delimiter)</code>	String	<code>st</code> : the input string. <code>delimiter</code> : the delimiter to find inside the input string.	Tries to find the delimiter passed in the second argument inside the string in the first argument. If found, it returns the string composed of the characters to the right of the delimiter in the input string.
<code>toString(integer int, string format)</code>	String	<code>int</code> : the input integer. <code>format</code> : indicates the type of format for the resulting string.	Returns a string formatted according to <code>format</code> argument. <code>format</code> possible values: <ul style="list-style-type: none">• "MB": Mbytes, using 2 decimals digits precision. Indicate <code>int</code> argument in kilobytes.• "GB": Gbytes, using 2 decimals digits precision. Indicate <code>int</code> argument in kilobytes.

(1) `elem` can be a numeric value, an IPv4 or IPv6 address or prefix, or a list or a range of integers or IP addresses values.

6.2.2.3.1 Format for Periods in Function `inPeriod`

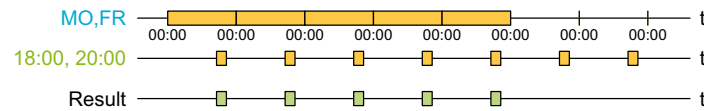
The second parameter of the `inPeriod` function is composed of expressions referring to date and time periods.

Examples:

— `inPeriod(now.offset, "MO,FR/18:00,20:00")`



The function returns true in the period between Monday and Friday and within that days only from 18:00:00 to 20:00:59.



— `inPeriod(now.offset, strcat("02-06-2012,",Subscriber.date1FU))`

being `Subscriber.date1FU` an operator-specific attribute provisioned in the Subscriber profile) containing a date and time string according to `dd-mm-yyyy[Thh:mm:ss]` format.

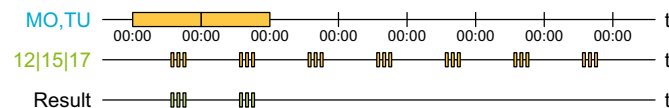
The function returns true in the period between 2nd of June 2012 and the date specified in Subscriber profile `date1FU` attribute.

— `inPeriod(now.offset, "01-07-2011T12:00:00,31-07-2011T13:00:00")`

The function returns true in the period between July the 1st, 2011 at 12:00:00 and July the 31st, 2011 at 13:00:00.

— `inPeriod(now.offset, "MO,TU/12|15|17")`

Period between Monday and Tuesday only from 12:00:00 to 12:59:59, from 15:00:00 to 15:59:59 and from 17:00:00 to 17:59:59.



— `inPeriod(now.offset, "MO|WE|FR")`

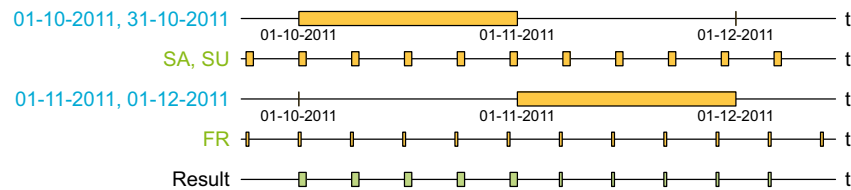
Every monday, wednesday, and friday.

— `inPeriod(now.offset, "01-10-2011/SA,SU")`

October the 1st, 2011 only if it is weekend (saturday or sunday).

— `inPeriod(now.offset, "01-10-2011,31-10-2011/SA,SU;01-11-2011,31-11-2011/FR")`

returns true for every weekend from October the 1st, 2011 until October 31, 2011 or for every friday from November the 1st, 2011 until November 31, 2011.



— `inPeriod(now.offset, "M0")`

returns true every Monday, starting at 00:00:00 and ending at 23:59:59.

— `inPeriod(now.offset, "01-07-2011T12:00")`

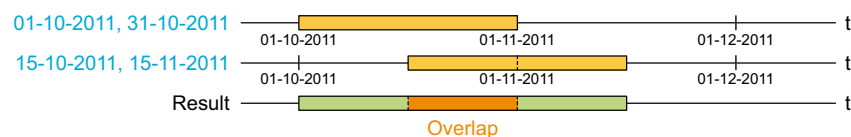
The day, month, year, hours, and minutes are specified but the seconds are not, so the period covers the whole minute: so "01-07-2011T12:00" is equivalent to "01-07-2011T12:00:00,01-07-2011T12:00:59".

— `inPeriod(now.offset, "10")`

The hour is specified but the minutes and seconds are not, so the period covers the whole hour: so "10" is equivalent to "10:00:00,10:59:59".

— `inPeriod(now.offset, "01-10-2011,31-10-2011;15-10-2011,15-11-2011")`

From October the 1st, 2011 to October 31, 2011 or from October 15, 2011 to November 15, 2011. The two periods overlap and therefore the validity returned by the function could be not precise.



— `inPeriod(now.offset, "01-12-2011,31-12-2011/M0,FR/08|14|22:30,23:59")`

Returns true for December only for days from monday to friday AND for hours 8:00:00 until 8:59:59 OR from 14:00:00 until 14:59:59 OR from 22:30:00 until 23:59:59.

It is shorter to be configured in this way than its equivalent

"01-12-2011,31-12-2011/M0|TU|WE|TH|FR/08:00:00,08:59:59|14:00:00,14:59:59|22:30:00,23:59:59"

The language of the periods according to Backus-Naur Form (BNF) specification (refer to Backus Naur Form - RFC 2234) has the following format:

`<periods> ::= <period> [";"<periods>]`



```

<period> ::=
    <subperiodDate>["/"<subperiodWeek>]["/"<subperiodTime>] |
    <subperiodWeek>["/"<subperiodTime>] |
    <subperiodTime>

<subperiodDate> ::= <subperiodDateItem>["|"<subperiodDate>]
<subperiodWeek> ::= <subperiodWeekItem>["|"<subperiodWeek>]
<subperiodTime> ::= <subperiodTimeItem>["|"<subperiodTime>]

<subperiodDateItem> ::= <startInDate> ["," <endInDate>]
<subperiodWeekItem> ::= <startInWeek> ["," <endInWeek>]
<subperiodTimeItem> ::= <startInTime> ["," <endInTime>]

<startInDate> ::= <dd>"-"<mm>"-"<yyyy>["T"<hh>[":"<mm>[":"<ss>]]]
<endInDate> ::= <dd>"-"<mm>"-"<yyyy>["T"<hh>[":"<mm>[":"<ss>]]]

<startInWeek> ::= "MO" | "TU" | "WE" | "TH" | "FR" | "SA" | "SU"
<endInWeek> ::= "MO" | "TU" | "WE" | "TH" | "FR" | "SA" | "SU"

<startInTime> ::= <hh>[":"<mm>[":"<ss>]]
<endInTime> ::= <hh>[":"<mm>[":"<ss>]]

```

The meaning of the period elements is the following:

- `period` represents each period that is going to be considered.

The character “;” can be used to specify a list of different periods; the function returns true in case any of the specified periods is satisfied (logical OR).

Note: When several periods are passed as second argument, Ericsson recommends avoiding that the defined periods overlap. If the periods overlap, the precision of the validity returned by the function `inPeriod` could be not precise.

A period is composed of at least one of the following elements:

- `subperiodDateItem` represents a date period. It has a beginning specifying when it starts and optionally an end.
- `subperiodWeekItem` represents a period specified with weekdays (from Monday to Sunday). It has a beginning specifying the day when it starts and optionally an end. If the end is not specified, the period ends at the end (23:59:59) of the specified starting day.
- `subperiodTimeItem` represents a time period. It has a beginning specifying the time when it starts and optionally an end.

Inside these elements, the beginning and the end of the period are separated by the character “,” (range).



Note: : For subperiodWeekItem and subperiodTimeItem, if the end period is specified and is previous than the starting period, the function returns false. So, in the following examples, the function returns false.

```
inPeriod(now.offset, "FR,TU")  
inPeriod(now.offset, "18:00:00,08:00:00")
```

Each of these elements can appear more than once in one subPeriod, using character “|” (logical OR).

To make configuration easier (and shorter to be written), the character “/” can be used for common factor formula (logical AND), together with ranges (“,”) and logical OR (“|”).

Precedence among “,” “/”, and “|”: considering the periods as a mathematical logical expression, when several of the special characters appear, “,” has higher priority, then “/” and as last, “|” has the lowest priority.

6.2.2.3.2 Examples of Function in Range

```
inRange(AccessData.host.version, "4000-6000, 8000-9000")  
inRange(AccessData.subscriber.locationInfo.sgsnAddress, "160.50.40.0/24,160.50.40.50-160.50.40.60")  
inRange(AccessData.subscriber.locationInfo.sgsnAddress, "2001:0DB8:0::1428:57AB/32")  
inRange(AccessData.subscriber.locationInfo.sgsnAddress, "2001:0DB8:0000:0000:0000:0000:1428:57AB-2001:0DB8::FFFF:FFFF")  
inRange(TdfData.serverIp, "160.50.1.1-160.50.255.255")  
inRange(TdfData.serverPort, "5000-5100,9500-9999")
```



Reference List

Standards

- [1] Backus Naur Form - RFC 2234