CHAPTER 2

Concepts and Terms

This module describes various terms and concepts that are utilized when working with the SCMS SCE Subscriber API.

Subscriber Characteristics

One of the fundamental entities in the Service Control Application for Broadband (SCA BB) solution is a subscriber . A subscriber is the entity that the SCA BB solution individually monitors, accounts, and enforces a service configuration. The following sections briefly describe the characteristics of the subscriber in the SCA BB. For more information about the format and usage of the subscriber's characteristics, see the Getting Familiar with the API Data Types, page 4-1module.

Subscriber ID

Subscriber ID is a subscriber unique identifier, for example, a user name, IMSI (International Mobile Subscriber Identity), or other codes that uniquely identify a subscriber.

Anonymous Subscriber ID

When working in the Pull Model integration, the SCE assigns each unknown subscriber IP address with a temporary Subscriber ID, Anonymous Subscriber ID, until it receives the real Subscriber ID from the Policy Server.

For more information on the Pull Model integration, see the Information About Subscriber Integration Models section.

Network ID

The SCE correlates a certain traffic flow to a subscriber by mapping a network identifier, for example, IP address, IP range, or VLAN, to the subscriber entity.

Policy Profile

A Policy Profile includes a set of parameters used by the SCA BB solution to define what policy is enforced on the subscriber.

Quota

A quota includes the quota-bucket values of the service quota or quotas available for the usage of the subscriber.

Information About Subscriber Integration Models

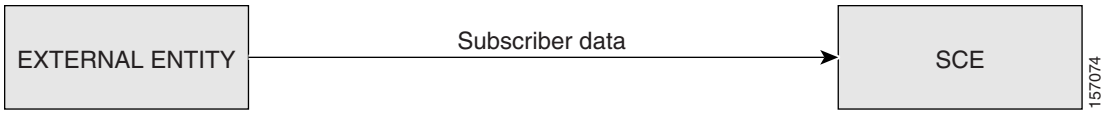
The following terms describe two models of a dynamic subscriber integration that the SCE platform supports.

- Push Model, page 2-3
- Pull Model, page 2-3

Push Model

In push model integration, an external server introduces (pushes) the subscribers to the SCE platform. This is performed whenever a new subscriber logs in to the network or the external server presumes to know all subscribers and introduces them to the SCE box when they connect.

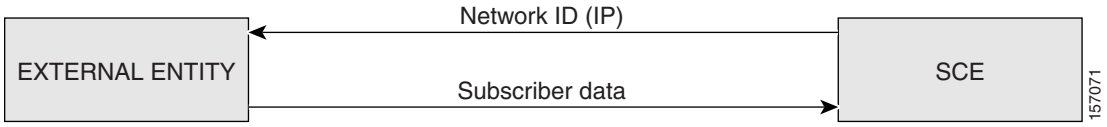
Figure 2-1 Push Model Schematic



Pull Model

In pull model integration, the SCE platform requests subscriber data from the external entity when it encounters traffic of an unknown subscriber, known as an anonymous subscriber. The external entity retrieves the required subscriber information and sends it back to the SCE platform.

Figure 2-2 Pull Model Schematic



Non-blocking Model

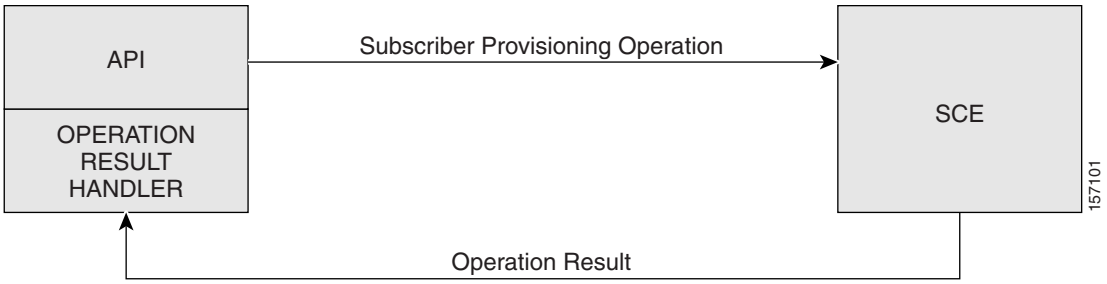
The SCE Subscriber API is implemented using a non-blocking model. Non-blocking methods return immediately, even before the completion of a subscriber provisioning operation. The Non-blocking Model method is advantageous when the operation is lengthy and involves I/O. Performing the operation in a separate thread allows the caller to continue doing other tasks and it improves overall system performance.

The operation results are either returned to an Observer object (Listener) or may not be returned at all.

The API supports retrieval of operation results using an operation result handler described in the Information About Result Handling section.

The following diagram illustrates the Non-blocking Model method during a subscriber provisioning operation:

Figure 2-3 Non-blocking Model

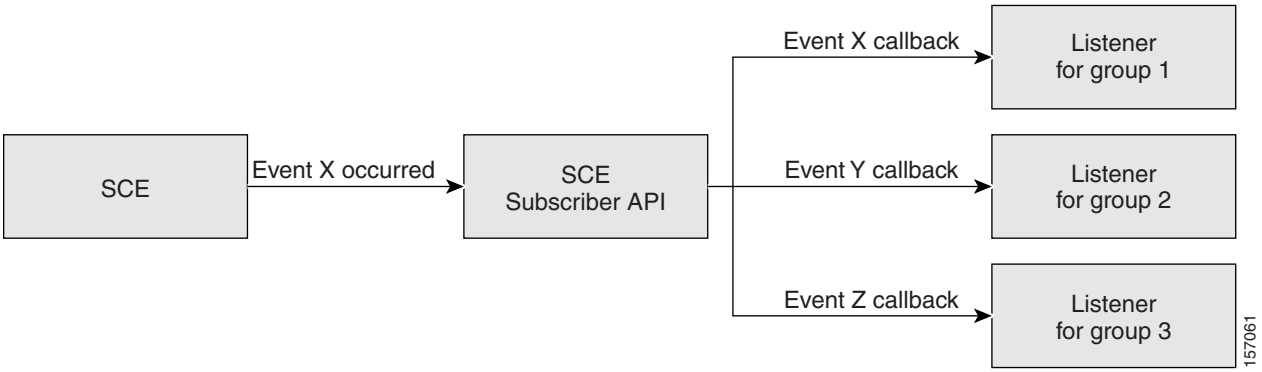


Operation results can be used for operation result error logging or for inspection of the parameters used by the operation.

Indications Listeners

The API provides the user with the ability to receive an indication when certain events occur on the SCE platform. The API dispatches the indications received from the SCE to the interested entities, called listeners , by activating the relevant Listener's callback methods. The indications are separated into several logical groups when only **one** listener can be defined for each group of indications.

Figure 2-4 Indication Listeners



To receive certain indications, you need to register a listener to the API that implements the required callback functions. After the listener is registered, the API can dispatch the required indications to the listener. The SCMS SCE Subscriber API provides three types of indications when separate listeners are registered to the following types of the indications:

- Login-pull indications
- Logout indications
- Quota indications

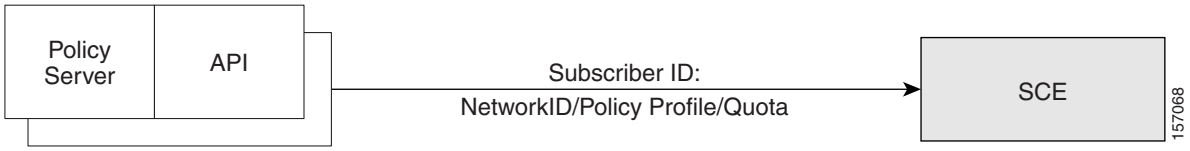
For more information about listener indications, see the API Eventsmodule.

Supported Topologies

The following topologies are recommended to use with the SCMS SCE Subscriber API:

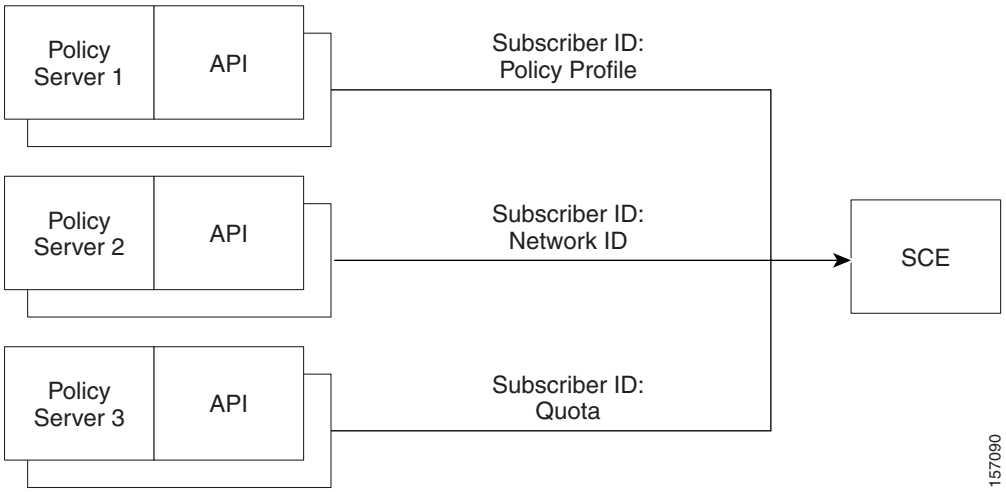
- One policy server (or two-node cluster) that is responsible for all aspects of the subscriber provisioning process:

Figure 2-5 Supported Topologies - One Policy Server



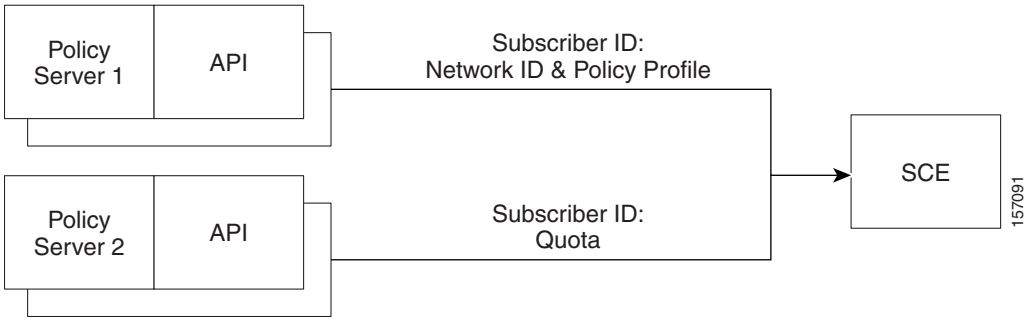
- Three policy servers (or three two-node clusters)—Every server is responsible for a different aspect of the subscriber provisioning process:

Figure 2-6 Supported Topologies - Three Policy Servers



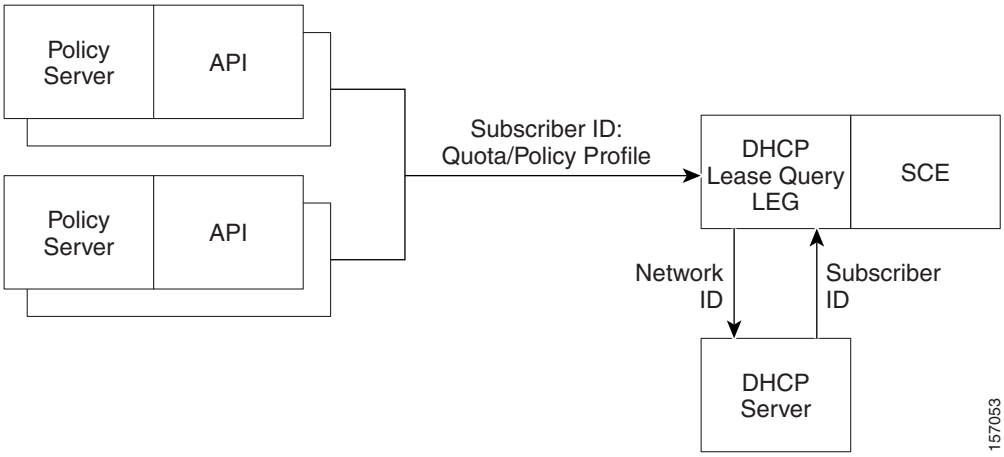
- Two policy servers (or two two-node clusters) when one of the servers is responsible for two aspects of the subscriber provisioning and the other server is responsible for one aspect only (any combination is allowed). For example:

Figure 2-7 Supported Topologies - Two Policy Servers



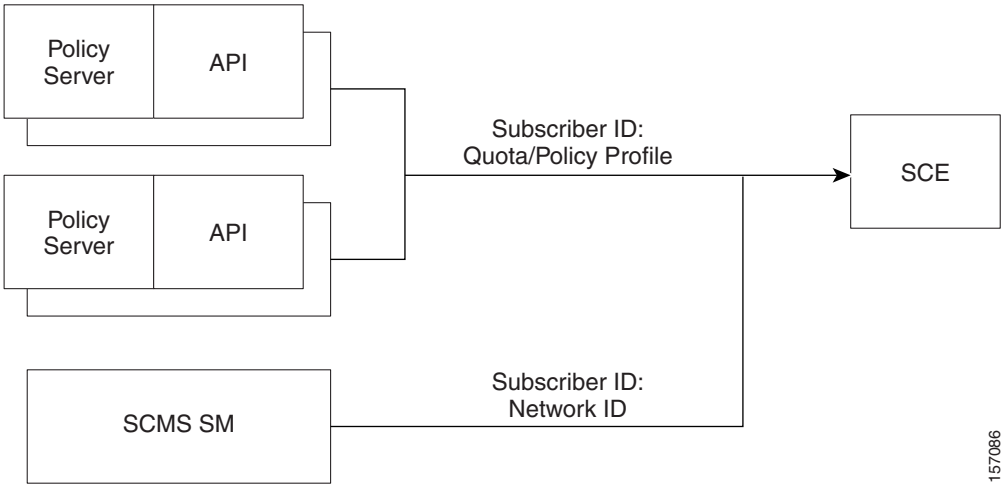
- DHCP Lease Query LEG, which is responsible for mapping a Network ID to a Subscriber ID, with one or more policy servers as described in the three policy server diagram above. The following diagram shows the DHCP Lease Query LEG:

Figure 2-8 Supported Topologies - DHCP Lease Query LEG



- SCMS SM, which is responsible for mapping Network ID to Subscriber ID, with one or more policy servers. The number of policy servers depends on whether the SM is used for policy profile provisioning in addition to the network ID:

Figure 2-9 Supported Topologies - SM



Note The API itself does not limit the use of any topology; however, the SCE platform does not correlate between all the entries (Policy Servers) that perform subscriber provisioning. Therefore you should be **extremely** careful when using more than one Policy Server for the **same provisioning purpose** (for example Network ID/Subscriber ID correlation). If you are not careful when using more than one Policy Server, the SCE platform may receive different information for the same subscriber from the two policy servers responsible for the same aspect of the subscriber provisioning. This may cause a loss of synchronization with at least one policy server. For example, using two policy servers that are responsible for providing Subscriber ID/Network ID correlation for the same subscriber will produce the situation where the SCE is always synchronized with the policy server that performed the last update for this subscriber.

Multi-threading Support

The API supports an unlimited number (limited by the available memory) of threads calling its methods simultaneously.

Note In a multi-threaded scenario, the order of invocation is **guaranteed**: the API performs operations in the same chronological order that they were called.

Auto-reconnect Support

The API supports auto-reconnection to the SCE in case of connection failure. When this option is activated, the API can determine when the connection to the SCE is lost. When the connection is lost, the API activates a reconnection task that tries to reconnect to the SCE again in a configurable interval time until reconnection is successful.

Reliability Support

The SCMS SCE Subscriber API is implemented as a *reliable* API. The API ensures that no requests to the SCE are lost and no indication from the SCE is lost. The API maintains an internal storage for all API requests that were sent to the SCE. Only after receiving an acknowledgement from the SCE that the request was handled, it considers the request as **committed** and the API can remove the request from its internal storage. If a connection failure occurs between the API and the SCE, the API accumulates all requests in its internal storage until the connection to the SCE is reestablished. On reconnection, the API resends all **non-committed** requests to the SCE, ensuring that no requests are lost.

Note The order of resending requests is **guaranteed** : the API resends the requests in the same chronological order that they were called.

High Availability Support

The API provides high availability support. It assumes that the high availability scheme of the policy server is a two-node cluster type where only one server is active at any given time. The other server, in standby, is not connected to the SCE. For more information, see the Implementing High Availability, page 5-41.

Synchronization

The SCE and Policy Server must bekept synchronized concerning the subscribers for which the SCE is handling their internal parameters. Otherwise, the SCE might confuse one of the subscriber's traffic to another subscriber, or the subscriber's SLA (Service Level Agreement) will not be enforced because of a change in the policy that did not reach the SCE. For more information, see Information About SCE-API Synchronization, page 5-35.

Practical Tips

When implementing the code that integrates the API with your application you should consider the following practical tips:

- Connect once to the SCE and maintain an open API connection to the SCE at all times, using the API many times. Establishing a connection is a timely procedure, which allocates resources on the SCE side and the API client side.
- Share the API connection between your threads - it is better to have one connection per Policy Server. Multiple connections require more resources on the SCE and client side.
- Do not implement synchronization of the calls to the API. The client automatically synchronizes calls to the API.
- If the Policy Server application has bursts of logon operations, enlarge the internal buffer size accordingly to hold these bursts (Non-Blocking flavor).
- During the integration, use the logging capabilities that are described in the SCE Loggingand the API Client Logging sections to view the API operations in the SCE's client logs and to troubleshoot problems during the integration, if any.
- Use the debug mode for the Policy Server application that logs/prints the return values of the operations.
- Use the automatic reconnect feature to improve the resiliency of the connection to the SCE.