CHAPTER 2

General API Concepts

Revised: December 21, 2012, OL-26795-02

Introduction

This chapter describes the concepts that pertain to working with the Cisco Service Control Subscriber Manager Java API.

This chapter consists of the following sections:

Blocking API and the Nonblocking API

This section describes the differences between the Blocking API and the Nonblocking API.

Blocking API

In a Blocking API operation, which is the most common, every method returns *after* its operation is performed.

The Cisco Service Control Subscriber Manager Blocking Java API provides a wide range of operations. It contains most of the functionality of the Nonblocking API and many functions that the Nonblocking API does not provide. The Blocking API does not support reliability and autoreconnect functionality.

Nonblocking API

In a Nonblocking Java API operation, every method returns *immediately*, even before the completion of its operation. The operation results are either returned to an observer object (listener) or not returned at all.

The Nonblocking API method is preferred when the operation is lengthy and involves I/O. When the operation is performed in a separate thread, the calling program can continue performing other tasks, which improves the overall system performance.

The Cisco Service Control Subscriber Manager Nonblocking Java API contains a small number of nonblocking operations. The API supports the retrieval of operation results by using a result listener.

The Cisco Service Control Subscriber Manager Nonblocking Java API supports two modes—reliable and nonreliable. For more information about the reliability modes, see the "Reliability Support" section on page 4-2.

API Initialization

To initialize the API, you must complete three tasks:

- Construct the API by using one of its constructors. For details, see the "API Construction" section on page 2-3.
- Perform the API-specific setup operations. For details, see the "Setup Operations" section on page 2-3.
- Connect the API to the Cisco Service Control Subscriber Manager. For details, see the "Connecting to the Cisco Service Control Subscriber Manager" section on page 2-4.

API Construction

Blocking and Nonblocking APIs have two constructors:

- An Empty constructor
- A Constructor that accepts a Login Event Generator (LEG) name as a parameter.

Constructor that Accepts a LEG Name

Set the LEG name if you intend to turn on the Cisco Service Control Subscriber Manager LEG failure processing options in the Cisco Service Control Subscriber Manager. For more information about the LEG software components and SM-LEG failure handling, see the *Cisco Service Control Management Suite Subscriber Manager User Guide*.

The Cisco Service Control Subscriber Manager uses the LEG name when recovering from a connection failure. A constant string that identifies the API is appended to the LEG name as follows:

- For blocking API—.B.SM-API.J
- For nonblocking API—.NB.SM-API.J

Blocking API—Example

If the provided LEG name is my-leg.10.1.12.45-version-1.0, the actual LEG name is my-leg.10.1.12.45-version-1.0.B.SM-API.J.

If no name is set, the LEG uses the hostname of the machine as the prefix of the name.

Additional constructors are available for the Nonblocking API. For more information, see the "Nonblocking API Construction" section on page 4-6.

Setup Operations

The setup operations differ for the two APIs. Both APIs support setting a disconnect listener, which is described in detail in the "DisconnectListener Interface" section on page 2-10.

Blocking API Setup

To set up the Blocking API, you must set an operation timeout value. For more information, see Chapter 3, "Blocking API".

Nonblocking API Setup

To set up the Nonblocking API, you must set a disconnect listener. For more details, see Chapter 4, "Nonblocking API".

Connecting to the Cisco Service Control Subscriber Manager

To connect to the Cisco Service Control Subscriber Manager, use one of the following connect methods.

- Use the following method to connect to the Cisco Service Control Subscriber Manager by using the default remote procedure call (RPC) TCP port (14374):

```
connect(String host)
```

- Use the following method to allow the caller to set the TCP port to which the API connects:

```
connect(String host, int port)
```

For both methods, the host parameter can be either an IP address or a reachable hostname.

At any time during the API operation, you can check if the API is connected by using the isConnected method.

API Finalization

To free the resources of both the server and the client, use the disconnect method.

Use a finally statement in your main class, as follows:

```
public static void main(String [] args) throws Exception {
    SMNonBlockingApi smnbapi = new SMNonBlockingApi();
    try {
        ...
    }
    finally {
        smnbapi.disconnect();
    }
}
```

Subscriber Name Format

Most methods of both APIs require the subscriber name as an input parameter.

A subscriber name can contain up to 64 characters. You can use all printable characters with an ASCII code between 32 and 126 (inclusive), except for 34 ("), 39 ('), and 96 (`).

Network ID Mappings

A network ID mapping is a network identifier that the Service Control Engine (SCE) device can map to a specific subscriber record. A typical example of a network ID mapping (or simply, mapping) is an IP address. Currently, the Cisco Service Control solution supports IP address, IP range, private IP address over VPN, private IP range over VPN, and VLAN mappings.

Both Blocking and Nonblocking APIs contain operations that accept mappings as a parameter, such as the following:

- The addSubscriber operation (Blocking API)
- The login method (Blocking or Nonblocking API)

When passing mappings to an API method, the caller is requested to provide two parameters:

- The java.lang.String mapping identifier or array of mapping types
- The short mapping type or array of mapping types

When passing arrays, the mappingTypes array must contain either the same number of elements as the mappings array or a single element. If the mappingTypes array contains a single element, all the mappings have the same type, specified by this single element.

Note Only one mapping type is allowed in a call to any of the APIs.

APIs supports the following subscriber mapping types:

- IP addresses or IP ranges
- (From Cisco Service Control Subscriber Manager, Release 3.8.5) IPv6 address
- Private IP addresses or private IP ranges over VPN
- VLAN tags
- (From Cisco Service Control Subscriber Manager, Release 3.8.5) IPv6 addresses or IPv6 prefix

For additional information, see the *Cisco Service Control Management Suite Subscriber Manager User Guide*.

Specifying IP Address Mapping

The string format of an IP address is the commonly used decimal notation:

```
IP-Address=[0-255].[0-255].[0-255].[0-255]
```

Example:

- 216.109.118.66

The mapping type of an IP address is provided in the com.pcube.management.api.SMApiConstants interface:

- com.pcube.management.api.SMApiConstants.MAPPING_TYPE_IP specifies a single IP mapping that matches the mapping identifier with the same index in the mapping identifier array.
- com.pcube.management.api.SMApiConstants.ALL_IP_MAPPINGS specifies that all the entries in the mapping identifier array are IP mappings.

Specifying IPv6 Address Mapping

The string format of an IP address is the commonly used hexadecimal colon-separated notation:

```
IPV6-Address=[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000
-FFFF]:[0000-FFFF]/[32-64]
```

Note The prefix value of all IPv6 addresses should be between 32 and 64. Only the first 64-bit of the IPv6 address is considered.

The mapping type of an IP address is provided in the com.pcube.management.api.SMApiConstants interface:

- com.pcube.management.api.SMApiConstants.MAPPING_TYPE_IPV6 specifies a single IP mapping that matches the mapping identifier with the same index in the mapping identifier array.
- com.pcube.management.api.SMApiConstants.ALL_IPV6_MAPPINGS specifies that all entries in the mapping identifier array are IPv6 mappings.

Example:

- 2001:db8:85a3::8a2e:370:7334/64

Specifying IP Range Mapping

The string format of an IP range is an IPv4 address in decimal notation and a decimal specifying the number of 1s in a bit mask:

```
IP-Range=[0-255].[0-255].[0-255].[0-255]/[0-32]
```

Examples:

- 10.1.1.10/32 is an IP range with a complete mask, that is, a regular IP address.
- 10.1.1.0/24 is an IP range with a 24-bit mask, that is, all the addresses ranging between 10.1.1.0 and 10.1.1.255.

Note The mapping type of an IP range is identical to the mapping type of the IP address.

Specifying Private IP Address or Private IP Range over VPN Mapping

The string format of an IPv4 address and an IP range are described in the "Specifying IP Address Mapping" section on page 2-6 and in the "Specifying IP Range Mapping" section on page 2-7. When the network ID mapping uses an IP address or range over VPN, the string format includes the VPN name.

Examples:

- 10.1.1.10@VPN1 is an IP address over the VPN named VPN1.
- 10.1.1.0/24@VPN2 is an IP range with a 24-bit mask, that is, all of the addresses in the range from 10.1.1.0 to 10.1.1.255 over the VPN named VPN2.

Note The mapping type of an IP address or IP range over VPN is identical to the mapping type of the IP address.

Specifying VLAN Tag Mapping

The string format for VLAN tag mapping is VLAN-tag = 0 to 4095.

The value is a decimal in the specified range.

The com.pcube.management.api.SMApiConstants interface also provides the mapping type:

- com.pcube.management.api.SMApiConstants.MAPPING_TYPE_VPN specifies a single VLAN mapping that matches the mapping identifier with the same index in the mapping identifier array.
- com.pcube.management.api.SMApiConstants.ALL_VPN_MAPPINGS specifies that all the entries in the mapping identifiers array are VLAN mappings.

Note The SMApiConstants.TYPE_VLAN and SMApiConstants.ALL_VLAN_MAPPINGS constants are deprecated. You should use the SMApiConstants.TYPE_VPN and SMApiConstants.ALL_VPN_MAPPINGS constants instead.

Subscriber Domains

The *Cisco Service Control Management Suite Subscriber Manager User Guide* defines the domain concept. Briefly, a domain identifies for the Cisco Service Control Subscriber Manager the Cisco SCE devices to which it updates subscriber records.

A domain name is a type of String. During system installation, the network administrator determines the system domain names, which, therefore, vary between installations. The APIs include methods that specify the domain to which a subscriber belongs. The APIs also provide methods that enable queries about system domain names. If an API operation specifies a domain name that does not exist in the Cisco Service Control Subscriber Manager domain repository, it is considered an error and RpcErrorException is returned.

The Cisco Service Control Subscriber Manager automatic domain roaming feature enables you to move subscribers between domains by calling the login method for a subscriber with an updated domain parameter as described in the "login" section on page 3-5.

Note Automatic domain roaming is not backward compatible with previous versions of the Cisco Service Control Subscriber Manager API. Previous versions did not permit changing the domain of the subscriber.

Subscriber Properties

Several operations manipulate subscriber properties. A subscriber property is a key/value pair that affects the way the Cisco SCE analyzes and reacts to network traffic generated by the subscriber.

For information about properties, see the *Cisco Service Control Management Suite Subscriber Manager User Guide* and the *Cisco Service Control Application for Broadband User Guide*. The latter provides application-specific information; it lists the subscriber properties that exist in the application running on your system, the allowed value set, and the significance of each property value.

To format subscriber properties for Java API operations, use the String arrays propertyKeys and propertyValues.

Note The arrays must be of the *same length*, and NULL entries are forbidden. Each key in the keys array has a matching entry in the values array; the value for propertyKeys[j] resides in propertyValues[j]. The mapping type of an IP range is identical to the mapping type of the IP address.

Example:

If the property keys array is {"packageId","monitor"} and the property values array is {"5","1"}, the properties are packageId=5, monitor=1.

Custom Properties

Some operations manipulate custom properties. Custom properties are similar to subscriber properties, but do not affect how the SCE analyzes and manipulates the subscriber traffic. The application management modules use custom properties to store additional information for each subscriber.

To format custom properties, use the String arrays customPropertyKeys and customPropertyValues. This formatting is the same as in the subscriber properties formatting described in the "Subscriber Properties" section on page 2-9.

DisconnectListener Interface

Both APIs (Blocking and Nonblocking) enable you to set a disconnect listener. The disconnect listener is an interface with a single method:

```
public interface DisconnectListener {
    /**
     * called when the connection with the server is down.
     */
    public void connectionIsDown();
}
```

Implement this interface to be notified when the API is disconnected from the Cisco Service Control Subscriber Manager.

To set a disconnect listener, use the setDisconnectListener method.

DisconnectListener Interface Example

The following example is a basic implementation of a disconnect listener that prints a message to stdout and exits:

```
import com.pcube.management.framework.rpc.DisconnectListener;

public class MyDisconnectListener implements DisconnectListener {

    public void connectionIsDown(){
        System.out.println("Message: connection is down.");
        System.exit(0);
    }
}
```

Exceptions

The same Java class, com.pcube.management.framework.rpc.RpcErrorException, provides all of the functional errors of the Cisco Service Control Subscriber Manager Java API. This is contrary to the normal Java usage. This approach was chosen because of the cross-language nature of the Cisco Service Control Subscriber Manager API. It enables all Cisco Service Control Subscriber Manager API implementations (Java, C, and C++) to appear similar.

Each exception provides the following information:

- Unique error code (long)
- Informative message (java.lang.String)
- Server-side stack trace (java.lang.String)

The error code can be interpreted by using com.pcube.management.api.SMApiConstants. See Appendix A, "List of Error Codes" for details about error codes and their significance.

Note Several types of errors can occur *only* if you use the Blocking API. These are operational errors that are related to operation-timeout handling. These errors are described in Chapter 3, "Blocking API."

Practical Tips

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

- Connect to the Cisco Service Control Subscriber Manager once and maintain an open API connection to the Cisco Service Control Subscriber Manager at all times, using the API many times. Establishing a connection is a timely procedure, which allocates resources on the Cisco Service Control Subscriber Manager side and the API client side.
- Share the API connection between your threads. It is best to have one connection per LEG. Multiple connections require more resources on the Cisco Service Control Subscriber Manager side and client side.
- Do not implement synchronization of the calls to the API. The client automatically synchronizes calls to the API.
- Place the API clients (LEGs) in the same order as the Cisco Service Control Subscriber Manager machine processor number.
- If the LEG application has bursts of login operations, enlarge the internal buffer size accordingly to hold these bursts (Nonblocking API).
- During the integration, set the Cisco Service Control Subscriber Manager logon_logging_enabled configuration parameter to enable viewing the API operations in the Cisco Service Control Subscriber Manager log. Setting this parameter enables you to troubleshoot the integration, if any problems arise.
- Use the debug mode for the LEG application that logs the return values of the nonblocking operations.
- Use the automatic reconnect feature to improve the resiliency of the connection to the Cisco Service Control Subscriber Manager.
- In cluster setups, connect the API by using the virtual IP address of the cluster and not the management IP address of one of the machines.