CHAPTER 2

General API Concepts

Revised: July 23, 2009, OL-7204-09

Introduction

This chapter describes the various concepts that are used while working with the SM Java API.

Information About the Blocking API and the Nonblocking API

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

Blocking API

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

The SM 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 advantageous when the operation is lengthy and involves I/O. Performing the operation in a separate thread allows the calling program to continue doing other tasks and it improves overall system performance.

The SM Nonblocking Java API contains a small number of nonblocking operations. The API supports retrieval of operation results using a result listener.

The SM Nonblocking Java API supports two modes: reliable and nonreliable. For more information about the reliability modes, see Information About Reliability Support, page 4-1.

Information About API Initialization

There are three main steps to initialize the API:

- Construct the API using one of its constructors.
- Perform the API-specific setup operations.
- Connect the API to the SM.

The following sections describe these three steps.

Initialization examples can be found within the code examples sections under each API.

API Construction

Blocking and Nonblocking APIs have two common constructors:

- An empty constructor
- A constructor that accepts a LEG name as a parameter.

Constructor that Accepts a LEG Name

Set the LEG name if you intend to turn on the SM-LEG failure handling options in the SM. You should read about the LEG software components and SM-LEG failure handling in the *Cisco Service Control Management Suite Subscriber Manager User Guide*.

The SM will use the LEG name when recovering from a connection failure. A constant string that identifies the API will be appended to the LEG name as follows:

- For blocking API: **.B.SM-API.J**
- For nonblocking API: **.NB.SM-API.J**

Example (Blocking API)

If the provided LEG name is **my-leg.10.1.12.45-version-1.0**, the actual LEG name will be **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.

For additional information about LEG-SM failure handling, see the "Configuration File Options" chapter of the *Cisco Service Control Management Suite Subscriber Manager User Guide*.

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

Setup Operations

The setup operations differ for the two APIs. Both APIs support setting a disconnect listener, described in detail in the Information About the DisconnectListener Interface section.

- Blocking API Setup, page 2-3
- Nonblocking API Setup, page 2-3

Blocking API Setup

To set up the Blocking API, you need to set an operation timeout value. For more information, see Blocking API, page 3-1.

Nonblocking API Setup

To set up the Nonblocking API you are required to set a disconnect listener. For more details, see Nonblocking API, page 4-1.

Connecting to the Subscriber Manager

To connect to the SM, use one of the following **connect** methods.

- Use the following method to connect to the SM using the default 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 server and client, use the **disconnect** method.

It is recommended that you use a **finally** statement in your main class; for example:

```
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. This section lists the formatting rules of a subscriber name.

It can contain up to 64 characters. All printable characters with an ASCII code between 32 and 126 (inclusive) can be used; except for 34 ("), 39 ('), and 96 (`).

Information About Network ID Mappings

A network ID mapping is a network identifier that the SCE device can relate 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. Examples are:

- 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:

- A **java.lang.String** mapping identifier or array of mapping types
- A 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 mappings have the same type, specified by this single element.

The API supports the following subscriber mapping types:

- IP addresses or IP ranges
- Private IP addresses or private IP ranges over VPN
- VLAN tags

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]
```

Examples:

- 216.109.118.66
- The mapping type of an IP address is provided in the interface **com.pcube.management.api.SMApiConstants**:
  - **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 identifiers array are IP mappings.

Specifying IP Range Mapping

The string format of an IP range is an IP 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 full 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 IP address and an IP range are described in Specifying IP Address Mapping and Specifying IP Range Mapping. 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 ranging between **10.1.1.0** and **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 = 0-4095.

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 and it is recommended to use the **SMApiConstants.TYPE_VPN** and **SMApiConstants.ALL_VPN_MAPPINGS** constants instead.

Subscriber Domains

The *Cisco Service Control Management Suite Subscriber Manager User Guide* explains in detail the domain concept. Briefly, a domain is an identifier that tells the SM which SCE devices to update with the subscriber record.

A domain name is of type **String**. During system installation, the network administration determines the system domain names, which therefore vary between installations. The APIs include methods that specify to which domain a subscriber belongs and allow queries about the system's domain names. If an API operation specifies a domain name that does not exist in the SM domain repository, it is considered an error and an **RpcErrorException** will be returned.

The SM's Automatic Domain Roaming feature allows subscribers to be moved between domains by calling the login method for a subscriber with an updated domain parameter.

Note Automatic domain roaming is not backwards compatible with previous versions of the SM API, which did not allow 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 SCE analyzes and reacts to network traffic generated by the subscriber.

More information about properties can be found in the *Cisco Service Control Management Suite Subscriber Manager User Guide* and in the *Cisco Service Control Application for Broadband User Guide*. The application user guide 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 will be **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's 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**, the same as in formatting Subscriber Properties, page 2-7.

Information About the DisconnectListener Interface

Both APIs (Blocking and Nonblocking) allow setting 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 SM.

To set a disconnect listener, use the **setDisconnectListener** method.

DisconnectListener Interface Example

The following example is a simple 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 SM Java API. This is contrary to the normal Java usage. This approach was chosen because of the "cross-language" nature of the SM API. It allows all SM API implementations (Java, C, and C++) to look and feel the same.

Each exception provides the following information:

- A unique error code (**long**)
- An informative message (**java.lang.String**)
- A server-side stack trace (**java.lang.String**)

The error code can be interpreted using **com.pcube.management.api.SMApiConstants**. See the List of Error Codes for more details about error codes and their significance.

Note Several types of errors can occur **only** when the Blocking API is used. These are operational errors related to operation-timeout handling. They are described in detail in the Blocking API module.

Practical Tips

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

- Connect to the SM once and maintain an open API connection to the SM at all times, using the API many times. Establishing a connection is a timely procedure, which allocates resources on the SM side and the API client side.
- Share the API connection between your threads. It is better to have one connection per LEG. Multiple connections require more resources on the SM and client side.
- Do not implement synchronization of the calls to the API. The client automatically synchronizes calls to the API.
- It is recommended to place the API clients (LEGs) in the same order of the SM machine processor number.
- If the LEG application has bursts of logon operations, enlarge the internal buffer size accordingly to hold these bursts (Nonblocking flavor).

- During the integration, set the SM **logon_logging_enabled** configuration parameter to view the API operations in the SM log to troubleshoot the integration, if any problems arise.
- Use the debug mode for the LEG application that logs/prints the return values of the nonblocking operations.
- Use the automatic reconnect feature to improve the resiliency of the connection to the SM.
- In cluster setups, connect the API using the virtual IP address of the cluster and not the management IP address of one of the machines.