CONTENTS

About this Guide vii
  Document Revision History vii
  Organization viii
  Related Documentation viii
  Conventions viii

Obtaining Documentation and Submitting a Service Request ix

CHAPTER 1 1-1

Getting Started 1-1
  Information About the Java API 1-1
    Introduction 1-1
    Platforms 1-1
    Package Content 1-1

  How to Install the Java API 1-2
    Subscriber Manager Setup 1-2
    Installing on a UNIX Platform 1-3
    Installing on a Windows Platform 1-3

  Compiling and Running 1-3

CHAPTER 2 2-1

General API Concepts 2-1
  Information About the Blocking API and the Nonblocking API 2-1
    Blocking API 2-1
    Nonblocking API 2-2

  Information About API Initialization 2-2
    API Construction 2-2
      Constructor that Accepts a LEG Name 2-2
      Example (Blocking API) 2-2

    Setup Operations 2-3
      Blocking API Setup 2-3
      Nonblocking API Setup 2-3

    Connecting to the Subscriber Manager 2-3

  API Finalization 2-3

  Subscriber Name Format 2-4

  Information About Network ID Mappings 2-4
    Specifying IP Address Mapping 2-4
Specifying IP Range Mapping 2-5
Specifying Private IP Address or Private IP Range over VPN Mapping 2-5
Specifying VLAN Tag Mapping 2-5
Subscribers Domains 2-6
Subscribers Properties 2-6
Custom Properties 2-6
Information About the DisconnectListener Interface 2-7
   DisconnectListener Interface Example 2-7
Exceptions 2-7
Practical Tips 2-8

CHAPTER 3
Blocking API 3-1
Multithreading Support 3-1
ReplyTimeout and OperationTimeout Exception 3-2
Information About Blocking API Methods 3-2
   login 3-4
      Syntax 3-4
      Description 3-5
      Parameters 3-5
      RPC Exception Error Codes 3-5
      Return Value 3-6
      Examples 3-6
   logoutByName 3-7
      Syntax 3-7
      Description 3-7
      Parameters 3-7
      Return Value 3-7
      RPC Exception Error Codes 3-7
      Examples 3-8
   logoutByNameFromDomain 3-8
      Syntax 3-8
      Description 3-8
      Parameters 3-8
      Return Value 3-9
      RPC Exception Error Codes 3-9
      Example 3-9
   logoutByMapping 3-9
      Syntax 3-10
      Description 3-10
loginCable
Syntax: 3-11
Description: 3-11
Parameters: 3-11
Return Value: 3-12
RPC Exception Error Codes: 3-12
Examples: 3-12

logoutCable
Syntax: 3-12
Description: 3-13
Parameters: 3-13
Return Value: 3-13
RPC Exception Error Codes: 3-13
Examples: 3-13

addSubscriber
Syntax: 3-13
Description: 3-14
Example: 3-14
Parameters: 3-14
Return Value: 3-15
RPC Exception Error Codes: 3-15
Examples: 3-15

removeSubscriber
Syntax: 3-15
Description: 3-16
Parameters: 3-16
Return Value: 3-16
RPC Exception Error Codes: 3-16
Example: 3-16

removeAllSubscribers
Syntax: 3-16
Description: 3-16
Return Value: 3-17
RPC Exception Error Codes: 3-17

getNumberOfSubscribers
Syntax: 3-17
Contents

Description 3-17
Return Value 3-17
RPC Exception Error Codes 3-17
getNumberOfSubscribersInDomain 3-17
Syntax 3-18
Description 3-18
Parameters 3-18
Return Value 3-18
RPC Exception Error Codes 3-18
getSubscriber 3-18
Syntax 3-18
Description 3-18
Parameters 3-19
Return Value 3-19
RPC Exception Error Codes 3-19
Example 3-19
subscriberExists 3-19
Syntax 3-20
Description 3-20
Parameters 3-20
Return Value 3-20
RPC Exception Error Codes 3-20
subscriberLoggedIn 3-20
Syntax 3-20
Description 3-20
Parameters 3-21
Return Value 3-21
RPC Exception Error Codes 3-21
getSubscriberNameByMapping 3-21
Syntax 3-21
Description 3-21
Parameters 3-21
Return Value 3-22
RPC Exception Error Codes 3-22
getSubscriberNames 3-22
Syntax 3-22
Description 3-22
Parameters 3-23
Return Value 3-23
RPC Exception Error Codes 3-23
Example 3-23
getSubscriberNamesInDomain 3-24
Syntax 3-24
Description 3-24
Parameters 3-24
Return Value 3-24
RPC Exception Error Codes 3-24
getSubscriberNamesWithPrefix 3-24
Syntax 3-25
Description 3-25
Parameters 3-25
Return Value 3-25
RPC Exception Error Codes 3-25
getSubscriberNamesWithSuffix 3-25
Syntax 3-26
Description 3-26
Parameters 3-26
Return Value 3-26
RPC Exception Error Codes 3-26
getDomains 3-26
Syntax 3-26
Description 3-26
Return Value 3-27
RPC Exception Error Codes 3-27
setPropertiesToDefault 3-27
Syntax 3-27
Description 3-27
Parameters 3-27
Return Value 3-27
RPC Exception Error Codes 3-27
removeCustomProperties 3-28
Syntax 3-28
Description 3-28
Parameters 3-28
Return Value 3-28
RPC Exception Error Codes 3-28
Blocking API Code Examples 3-28
Getting Number of Subscribers 3-29
Adding a Subscriber, Printing Information, and Removing a Subscriber 3-29
CHAPTER 4

Nonblocking API  4-1

Information About Reliability Support  4-1
   Reliable Mode  4-1
   Nonreliable Mode  4-2

Autoreconnect Support  4-2
Multithreading Support  4-2

Information About the ResultHandler Interface  4-2
   ResultHandler Interface Example  4-3

Nonblocking API Construction  4-4
   Nonblocking API Syntax  4-4
   Nonblocking API Arguments  4-4
   Nonblocking API Examples  4-4

Nonblocking API Initialization  4-5
   Nonblocking API Initialization Syntax  4-5
   Nonblocking API Initialization Parameters  4-5
   Nonblocking API Initialization Example  4-5

Information About Nonblocking API Methods  4-6
   login  4-6
      Syntax  4-6
   logoutByName  4-7
      Syntax  4-7
   logoutByNameFromDomain  4-7
      Syntax  4-7
   logoutByMapping  4-7
      Syntax  4-7
   loginCable  4-7
      Syntax  4-7
   logoutCable  4-8
      Syntax  4-8

Nonblocking API Code Examples  4-8
   Login and Logout  4-8

APPENDIX A

List of Error Codes  A-1

   List of Error Codes  A-1
About this Guide

This document explains the Cisco Service Control Management Suite (SCMS) Subscriber Manager (SM) Java application programming interface (API).

The SCMS SM Java API is used for updating, querying, and configuring the subscriber manager (SM). It consists of two parts, which may be used separately or together without restriction:

1. SM Nonblocking Java API—A high-performance API with low visibility to errors and other operation results. Supports automatic integrations with OSS/AAA systems.

2. SM Blocking Java API—A more user-friendly API. Supports user interface applications for accessing and managing the SM.

A set of APIs with the same functionality is also available for the C/C++ environment.

This guide is for the networking or computer technician responsible for configuring the subscriber manager. It is also intended for the operator who manages Cisco Service Control Engine (SCE) platforms.

Document Revision History

The Document Revision History below records changes to this document.

<table>
<thead>
<tr>
<th>Revision</th>
<th>Cisco Service Control Release and Date</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL-7204-08</td>
<td>3.5.0 January, 2009</td>
<td>The SM Java API is operable on platforms that support version 5.0 and is no longer operable on platforms that support version 1.4.</td>
</tr>
<tr>
<td>OL-7204-07</td>
<td>3.1.6 May, 2008</td>
<td>Private IP address over VPN and private IP range over VPN are now supported for the Network ID. See Information About Network ID Mappings, page 2-4 and Specifying Private IP Address or Private IP Range over VPN Mapping, page 2-5.</td>
</tr>
</tbody>
</table>
About this Guide

Organization

This guide contains the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Getting Started, page 1-1</td>
<td>Describes the platforms on which the Java API can be used, and how to install, compile, and start running the Java API component.</td>
</tr>
<tr>
<td>2</td>
<td>General API Concepts, page 2-1</td>
<td>Describes various concepts that are used when working with the SM Java API.</td>
</tr>
<tr>
<td>3</td>
<td>Blocking API, page 3-1</td>
<td>Describes features and operation of the blocking API and provides code examples.</td>
</tr>
<tr>
<td>4</td>
<td>Nonblocking API, page 4-1</td>
<td>Describes features and operation of the nonblocking API and provides code examples.</td>
</tr>
<tr>
<td>A</td>
<td>List of Error Codes, page A-1</td>
<td>Lists error codes that are used in the Java API.</td>
</tr>
</tbody>
</table>

Related Documentation

Use this SCMS SM Java API Programmer Guide in conjunction with all of the SCMS Subscriber Manager user, API, and reference guides.

Conventions

This document uses the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>bold font</td>
<td>Commands and keywords and user-entered text appear in bold font.</td>
</tr>
</tbody>
</table>
Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly What’s New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:


Subscribe to the What’s New in Cisco Product Documentation as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS version 2.0.
Getting Started

This module describes the platforms on which the Java API can be used and how to install, compile, and start running the API.

- Information About the Java API, page 1-1
- How to Install the Java API, page 1-2
- Compiling and Running, page 1-3

Information About the Java API

- Introduction, page 1-1
- Platforms, page 1-1
- Package Content, page 1-1

Introduction

The Java API is used for updating, querying, and configuring the SCMS Subscriber Manager (SM). It consists of two parts, which can be used separately or together without restriction.

- SM Nonblocking Java API—A high-performance API with low visibility to errors and other operation results. It supports automatic integrations with OSS/AAA systems.
- SM Blocking Java API—A more user-friendly API. It supports user interface applications for accessing and managing the SM.

Platforms

The SM Java API was developed and tested on a Windows platform, but it is operable on any platform that supports Java version 5.0.

Package Content

For brevity, `<installdir>` refers to the installation directory `sm-java-api-vvv.bb`.
The `<installdir>/javadoc` folder contains the API JAVADOC documentation.
How to Install the Java API

The Java API distribution is part of the SCMS SM-LEG distribution file and is located in the SM_API directory.

The Java SM API is packaged in a UNIX tar file. You can extract the Java SM API using the UNIX tar utility or most Windows compression utilities.

- Subscriber Manager Setup, page 1-2
- Installing on a UNIX Platform, page 1-3
- Installing on a Windows Platform, page 1-3

Subscriber Manager Setup

The API connects to the PRPC server on the SM. For the API to work:

- The SM must be up and running, and reachable from the machine that hosts the API
- The PRPC server must be started.

The PRPC server is a proprietary RPC protocol designed by Cisco. For more information about the PRPC server, see the Cisco Service Control Management Suite Subscriber Manager User Guide.
Installing on a UNIX Platform

---
**Note**
The abbreviations *vvv* and *bb* stand for the Java SM API version and build number.
---
**Step 1**
Extract the SCMS SM-LEG distribution file.

**Step 2**
Locate the Java SM API distribution tar `sm-java-api-dist.tar`

**Step 3**
Extract the Java SM API distribution tar and obtain the `sm-java-api-vvv.bb.tar`

```
#> tar -xvf sm-java-api-dist.tar
```

**Step 4**
Extract the Java SM API package tar

```
#> tar -xvf sm-java-api-vvv.bb.tar
```

Installing on a Windows Platform

---
**Step 1**
Use a zip extractor (such as WinZip)

Compiling and Running

To compile and run a program that uses the SM Java API, `smapi.jar` must be in the CLASSPATH.

For example, if the program source is in `SMApiProgram.java`, use the following command line to compile the program:

```
#> javac -classpath smapi.jar SMApiProgram.java
```

After compiling the program, use the following command line to run the program:

```
#> java -cp .;<installdir>/lib/smapi.jar SMApiProgram
```
General API Concepts

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

- Information About the Blocking API and the Nonblocking API, page 2-1
- Information About API Initialization, page 2-2
- API Finalization, page 2-3
- Subscriber Name Format, page 2-4
- Information About Network ID Mappings, page 2-4
- Subscriber Domains, page 2-6
- Subscriber Properties, page 2-6
- Custom Properties, page 2-6
- Information About the DisconnectListener Interface, page 2-7
- Exceptions, page 2-7
- Practical Tips, page 2-8

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, page 2-1
- Nonblocking API, page 2-2

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):
  ```java
  connect(String host)
  ```
- Use the following method to allow the caller to set the TCP port to which the API connects:
  ```java
  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:

```java
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.
Information About Network ID Mappings

– `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 2-6.
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:

```java
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.

```java
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.
CHAPTER 3

Blocking API

This module introduces the Reply Timeout, a feature unique to the Blocking API.
The rest of the module lists all operations of the Blocking API, and provides code examples.

Note
If you only need to develop an automatic integration, skip this module and go directly to the Nonblocking API module.

- Multithreading Support, page 3-1
- ReplyTimeout and OperationTimeout Exception, page 3-2
- Information About Blocking API Methods, page 3-2
- Blocking API Code Examples, page 3-28

Multithreading Support

The Blocking API supports unlimited number of threads calling its methods simultaneously.

Note
In a multithreaded scenario for the Blocking API, the order of invocation is not guaranteed.

Example:
Thread-0 calls operation-0 at time-0, and thread-1 calls operation-1 at time-1, where time-1 is later than time-0. In this example, it is possible that operation-1 may be performed before operation-0, as shown in the following diagram (the vertical scale is time):
The SM allocates five threads to handle each API instance. It is recommended to develop a multithreaded application that uses the API with a number of threads in the order of the five threads. Implementing with more threads might result in longer delays for the calling threads.

### ReplyTimeout and OperationTimeout Exception

A blocking operation returns only when the operation result has been retrieved from the SM. If a networking malfunction or other error prevents the operation result from being retrieved, the caller will wait indefinitely. The SM API provides means of working around this situation.

The reply timeout feature (the `setReplyTimeout` method) lets the caller set a timeout. It will fire a `com.pcube.management.framework.rpc.OperationTimeoutException` when a reply does not return within the timeout period.

Calling the `setReplyTimeout` method with a long value sets a reply timeout. The reply timeout is interpreted in milliseconds. A zero value indicates that the operation should wait (freeze, hang) until a result arrives - or indefinitely, if no result arrives.

There is an alternative way to release a method call that is blocking the caller, who is waiting for a result to arrive: Call the `interrupt` method of the calling thread: a `java.lang.InterruptedException` will then be returned to the caller.

### Information About Blocking API Methods

This section lists the methods of the Blocking API. A description of each method's input parameters and return values follows the syntax of each method.

The Blocking API is a superset of the Nonblocking API. Except for differences in return values and result handling, identical operations in both APIs have the same functions and syntax structure.
All the methods throw a `java.lang.IllegalStateException` when called before a connection with the SM is established.

The Blocking API methods can be classified into the following categories:

- **Dynamic IP and property allocation**—For using the SM API for integration with an AAA system, the following methods are relevant. These methods are not designed to add or remove subscribers from the database, but to modify dynamic parameters (such as IP addresses) of existing subscribers:
  - login, page 3-4
  - logoutByName, page 3-7
  - logoutByMapping, page 3-9
  - loginCable, page 3-11
  - logoutCable, page 3-12

- **Static/Manual Subscriber configuration**—For example, for GUI usage, the following methods are relevant:
  - addSubscriber, page 3-13
  - removeSubscriber, page 3-15
  - removeAllSubscribers, page 3-16
  - setPropertiesToDefault, page 3-27
  - removeCustomProperties, page 3-28

- For simple read-only operations, performed independently on the subscriber awareness mode, the following methods are relevant:
  - getNumberOfSubscribers, page 3-17
  - getNumberOfSubscribersInDomain, page 3-17
  - getSubscriber, page 3-18
  - subscriberExists, page 3-19
  - subscriberLoggedIn, page 3-20
  - getSubscriberNameByMapping, page 3-21
  - getSubscriberNames, page 3-22
  - getSubscriberNamesInDomain, page 3-24
  - getSubscriberNamesWithPrefix, page 3-24
  - getSubscriberNamesWithSuffix, page 3-25
  - getDomains, page 3-26

It is possible to mix methods from different categories in a single application. The classification is presented for clarification purposes only.

- login, page 3-4
- logoutByName, page 3-7
- logoutByMapping, page 3-9
- loginCable, page 3-11
Information About Blocking API Methods

- logoutCable, page 3-12
- addSubscriber, page 3-13
- removeSubscriber, page 3-15
- removeAllSubscribers, page 3-16
- getNumberOfSubscribers, page 3-17
- getNumberOfSubscribersInDomain, page 3-17
- getSubscriber, page 3-18
- subscriberExists, page 3-19
- subscriberLoggedIn, page 3-20
- getSubscriberNameByMapping, page 3-21
- getSubscriberNames, page 3-22
- getSubscriberNamesInDomain, page 3-24
- getSubscriberNamesWithPrefix, page 3-24
- getSubscriberNamesWithSuffix, page 3-25
- getDomains, page 3-26
- setPropertiesToDefault, page 3-27
- removeCustomProperties, page 3-28

login

- Syntax, page 3-4
- Description, page 3-5
- Parameters, page 3-5
- RPC Exception Error Codes, page 3-5
- Return Value, page 3-6
- Examples, page 3-6

Syntax

```java
public void login(String subscriberName,
                  String[] mappings,
                  short[] mappingTypes,
                  String[] propertyKeys,
                  String[] propertyValues,
                  String domain,
                  boolean isMappingAdditive,
                  int autoLogoutTime)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```
Description

The login method adds or modifies a domain, mappings, and possibly properties of a subscriber that already exists in the SM database. It can be called with partial data; for example, with only mappings or only properties provided and NULL put in the unchanged fields.

If another subscriber with the same (or colliding) mappings already exists in the same domain, the colliding mappings will be removed from the other subscriber and assigned to the new subscriber.

If the subscriber does not exist in the SM database, it will be created with the data provided.

Parameters

subscriberName—See explanation of Subscriber Name Format, page 2-4.
mappings—See explanation of mappings and mapping types in the Information About Network ID Mappings section. If no mappings are specified, and the isMappingAdditive flag is TRUE, the previous mappings will be retained. If no such mappings exist, the operation will fail.
mappingTypes—See explanation of mappings and mapping types in the Information About Network ID Mappings section.
propertyKeys—See explanation of property keys and values in the Subscriber Properties section.
propertyValues—See explanation of property keys and values in the Subscriber Properties section.
domain—See explanation of Subscriber Domains, page 2-6.

If domain is NULL, but the subscriber already has a domain, the existing domain will be retained.

If the domain is different to the domain that was previously assigned to the subscriber, the subscriber will be removed automatically from the SCEs of the previous domain and moved to the SCEs of the new domain.

isMappingAdditive

- TRUE—Adds the mappings provided by this call to the subscriber record.
- FALSE— Overrides the mappings provided by this call with mappings that already exist in the subscriber record.

autoLogoutTime—Applies only to mappings provided as arguments to this method.

- Positive value (N)—Automatically logs out the mappings (similar to a logout method being called) after N seconds.
- 0 value—Maintains current expiration time for the given mappings.
- Negative value—Disables any expiration time that might have been set for the mappings given.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION
- ERROR_CODE_UNKNOWN

This error can be caused by the following:
- NULL value for domain parameter for the subscriber that does not exist or does not have a domain
- Invalid values for propertyValues parameter

For a description of error codes, see List of Error Codes, page A-1.

**Return Value**

None.

**Examples**

To add the IP address 192.168.12.5 to an existing subscriber named *john* without affecting existing mappings:

```
login(
  "john",                          // subscriber name
  new String[]{"192.168.12.5"},
  SMApiConstants.ALL_IP_MAPPINGS,
  null, null,
  "subscribers",                  // domain
  true,                           // isMappingAdditive is true
  -1);                            // autoLogoutTime set to infinite
```

To add the IP address 192.168.12.5 overriding previous mappings:

```
login(
  "john",                          // subscriber name
  new String[]{"192.168.12.5"},
  SMApiConstants.ALL_IP_MAPPINGS,
  null, null,
  "subscribers",                  // domain
  false,                          // isMappingAdditive is false
  -1);                            // autoLogoutTime set to infinite
```

To extend the auto logout time of 192.168.12.5 that was previously assigned to *john*:

```
login(
  "john",                          // the previously assigned IP
  new String[]{"192.168.12.5"},
  SMApiConstants.ALL_IP_MAPPINGS,
  null, null,
  "subscribers",                  // domain
  false,                          // isMappingAdditive
  300);                            // autoLogoutTime set to 300 seconds
```

To modify a dynamic property of *john* (e.g. package ID):

```
login(
  "john",
  null, null,
  new String[]{"packageId"},       // property key
  new String[]{"10"},              // property value
  "subscribers",                   // domain
  false, -1);
```

To add the IP address 192.168.12.5 to an existing subscriber named *john* without affecting existing mappings and modify a dynamic property of *john* (e.g. package ID):

```
login(
  "john",
  new String[]{"192.168.12.5"},
  SMApiConstants.ALL_IP_MAPPINGS,
  new String[]{"packageId"},       // property key
  new String[]{"10"},              // property value
  SMApiConstants.ALL_IP_MAPPINGS,
  null, null,
  "subscribers",                  // domain
  true,                           // isMappingAdditive is true
  -1);                            // autoLogoutTime set to infinite
```
logoutByName

- Syntax, page 3-7
- Description, page 3-7
- Parameters, page 3-7
- Return Value, page 3-7
- RPC Exception Error Codes, page 3-7
- Examples, page 3-8

Syntax

```java
public boolean logoutByName(String subscriberName,
                             String[] mappings,
                             short[] mappingTypes)
         throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Locates the subscriber in the database and removes mappings from it. If the subscriber does not exist, it does nothing.

Parameters

- `subscriberName`—See explanation of Subscriber Name Format, page 2-4.
- `mappings`—See explanation of mappings and mapping types in the Information About Network ID Mappings section. If no mappings are specified, all subscriber mappings will be removed.
- `mappingTypes`—See explanation of mappings and mapping types in the Information About Network ID Mappings section.

Return Value

- TRUE—If the subscriber was found and the subscriber's mappings were removed from the subscriber database.
- FALSE—If the subscriber was not found in the subscriber database.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
• ERROR_CODE_DOMAIN_NOT_FOUND
• ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
• ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Examples

To remove IP address 192.168.12.5 of subscriber john:

```java
boolean isExist = logoutByName("john",
    new String[]{"192.168.12.5"},
    SMApiConstants.ALL_IP_MAPPINGS);
```

To remove all IP addresses of subscriber john:

```java
boolean isExist = logoutByName("john", null, null);
```

logoutByNameFromDomain

• Syntax, page 3-8
• Description, page 3-8
• Parameters, page 3-8
• Return Value, page 3-9
• RPC Exception Error Codes, page 3-9
• Example, page 3-9

Syntax

```java
public boolean logoutByNameFromDomain(String subscriberName,
    String[] mappings,
    short[] mappingTypes,
    String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Similar to logoutByName, but also lets the caller provide the name of the domain to which the subscriber belongs. When the subscriber domain is known, use this method to get improved performance.

Parameters

subscriberName—See explanation of Subscriber Name Format, page 2-4.
mappings—See explanation of mappings and mapping types in the Information About Network ID Mappings section. If no mappings are specified, all the subscriber mappings will be removed
mappingTypes—See explanation of mappings and mapping types in the Information About Network ID Mappings section.
domain—See explanation of Subscriber Domains, page 2-6.
The operation will fail if either of the following conditions exists:
- The domain is null, but the subscriber exists in the database and belongs to a domain.
- The domain specified is incorrect.

**Return Value**
- TRUE—If the subscriber was found and removed from the subscriber database.
- FALSE—If the subscriber was not found in the subscriber database.

**RPC Exception Error Codes**

The following is the list of error codes that this method might return:
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

**Example**

To remove IP address 192.168.12.5 of subscriber john from domain subscribers:

```java
boolean isExist = logoutByNameFromDomain("john",
    new String[]{"192.168.12.5"},
    SMapiConstants.ALL_IP_MAPPINGS,
    "subscribers");

boolean isExist = logoutByNameFromDomain("john",
    null,
    null,
    "subscribers");
```

**logoutByMapping**

- Syntax, page 3-10
- Description, page 3-10
- Parameters, page 3-10
- Return Value, page 3-10
- RPC Exception Error Codes, page 3-10
- Example, page 3-10
Syntax

```java
public boolean logoutByMapping(String mapping,
        short mappingType,
        String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Locates a subscriber based on domain and mapping, and removes the mapping (the subscriber stays in the database).

Parameters

- `mapping`—See explanation of mappings and mapping types in the Information About Network ID Mappings section.
- `mappingType`—See explanation of mappings and mapping types in the Information About Network ID Mappings section.
- `domain`—See description in the Parameters section of the logoutByNameFromDomain operation.

Return Value

- TRUE—If the subscriber was found and removed from the subscriber database.
- FALSE—If the subscriber was not found in the subscriber database.

RPC Exception Error Codes

The following is the list of error codes that this method might return:
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Example

To remove IP address 192.168.12.5 from domain `subscribers`:

```java
boolean isExist = logoutByMapping(
        "192.168.12.5",
        SMApiConstants.MAPPING_TYPE_IP,
        "subscribers");
```
loginCable

- Syntax, page 3-11
- Description, page 3-11
- Parameters, page 3-11
- Return Value, page 3-12
- RPC Exception Error Codes, page 3-12
- Examples, page 3-12

Syntax

```java
public void loginCable(String CPE,
            String CM,
            String IP,
            int lease,
            String domain,
            String[] propertyKeys,
            String[] propertyValues)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

A login method adapted for the cable environment (calls the cable support module in the SM). This method is designed to log in CPEs and CMs to the SM. To log in a CPE, specify its CM MAC in the CM argument and the CPE MAC in the CPE argument. To log in a CM, specify the CM MAC address in both CPE and CM arguments. Note that the login of a CPE whose CM does not exist in the SM database will be ignored: the CM has to exist in the database, either by import or by a CM login operation. For additional information, see the CPE as Subscriber in Cable Environment chapter of the Cisco Service Control Management Suite Subscriber Manager User Guide.

**Note**
The name of the CPE in the SM database is the concatenation of the CPE and CM values with two underscore ‘_’ characters between them. The caller must make sure that the lengths of CPE and CM add up to no more than 38 characters.

Parameters

- **CPE**—A unique identifier of the CPE (usually a MAC address)
- **CM**—A unique identifier of the cable modem (usually a MAC address)
- **IP**—The CPE IP address
- **lease**—The CPE lease time
- **domain**—See explanation of Subscriber Domains, page 2-6. The domain will usually be CMTS IP.

**Note**
Domain aliases must be set on the SM in order for the CMTS IP to be correctly interpreted as a domain name. For information regarding aliases configuration see the “Default Domains Configuration” section of Cisco SCMS Subscriber Manager User Guide.
propertyKeys—See explanation of property keys and values in the Subscriber Properties section.
If the CPE is provided with partial or no application properties, the values for the missing application properties will be copied from the application properties of the CM to which this CPE belongs. Each CM application property thus serves as a default for the CPE under it.

propertyValues—See explanation of property keys and values in the Subscriber Properties section.

Return Value
None.

RPC Exception Error Codes
None.

Examples
To add the IP address 192.168.12.5 to a CM called CM1 with 2 hours lease time:

```
loginCable(
    "CM1",
    "CM1",
    "192.168.12.5",
    7200, // lease time in seconds
    "subscribers", null, null);
```

To add the IP address 192.168.12.50 to a CPE called CPE1 which is behind CM1 with lease time of 1 hours:

```
loginCable(
    "CPE1",
    "CM1",
    "192.168.12.50",
    3600, // lease time in seconds
    "subscribers", null, null);
```

logoutCable

- Syntax, page 3-12
- Description, page 3-13
- Parameters, page 3-13
- Return Value, page 3-13
- RPC Exception Error Codes, page 3-13
- Examples, page 3-13

Syntax

```
public boolean logoutCable(String CPE,
    String CM,
    String IP,
    String domain)
```
Description

Indicates a logout (CPE becoming offline) event to the SM cable support module.

Parameters

CPE—See description in the Parameters section of the loginCable method.
CM—See description in the Parameters section of the loginCable method.
IP—See description in the Parameters section of the loginCable method.
domain—See description in the Parameters section of the loginCable method.

Return Value

- TRUE—If the CPE was found and removed from the subscriber database.
- FALSE—If the CPE was not found in the subscriber database.

RPC Exception Error Codes

None.

Examples

To remove the IP address 192.168.12.5 from CPE1 which is behind CM1:

```java
boolean isExist = logoutCable(
   "CPE1",
   "CM1",
   "192.168.12.5",
   "subscribers");
```

addSubscriber

- Syntax, page 3-13
- Description, page 3-14
- Example, page 3-14
- Parameters, page 3-14
- Return Value, page 3-15
- RPC Exception Error Codes, page 3-15
- Examples, page 3-15

Syntax

```java
public void addSubscriber(String subscriberName,
   String[] mappings,
   short[] mappingTypes,
   String[] propertyKeys,
   String[] propertyValues,
```
String[] customPropertyKeys,
String[] customPropertyValues,
String domain
throws InterruptedException, OperationTimeoutException, RpcErrorException

Description

Creates a new subscriber record according to the method parameters and adds the record to the SM database.

If a subscriber by this name already exists, it will be removed before the new one is added. In contrast to login, which modifies fields and leaves unspecified fields unchanged, addSubscriber sets the subscriber exactly as specified by the parameters passed to it.

Note

It is recommended to call login method for existing subscribers, instead of addSubscriber. Dynamic mappings and properties should be set by using login. Static mappings and properties should be set at the first time the subscriber is created by using addSubscriber.

Note

With addSubscriber, the autologout feature is always disabled. To enable autologout, use login.

Example

Subscriber AB, already set up in the subscriber database, has a single IP mapping: IP1.

If an addSubscriber operation for AB is called with no mappings specified (NULL in both the mappings and mappingTypes fields), AB will be left with no mappings.

However, calling the login operation with these NULL-value parameters will not change AB’s mappings; AB will still have its previous IP mapping of IP1.

Parameters

subscriberName—See explanation of Subscriber Name Format, page 2-4.

mappings—See explanation of mappings and mapping types in the Information About Network ID Mappings section.

mappingTypes—See explanation of mappings and mapping types in the Information About Network ID Mappings section.

propertyKeys—See explanation of property keys and values in the Subscriber Properties section.

propertyValues—See explanation of property keys and values in the Subscriber Properties section.

customPropertyKeys—See explanation of custom property keys and values in the Custom Properties section.

customPropertyValues—See explanation of custom property keys and values in the Custom Properties section.

domain—See explanation of Subscriber Domains, page 2-6.

A NULL value indicates that the subscriber is domain-less.

If domain is NULL, but the subscriber already has a domain, the existing domain will be retained.
Return Value

None.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_UNKNOWN

This error code may indicate invalid values that were supplied for `propertyValues` parameter.

- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Examples

To add a new subscriber, `john`, with some custom properties:

```java
addSubscriber("john",
    null, null,   // dynamic mappings will be set by login
    null, null    // dynamic properties will be set by login
    new String[]{    // custom property keys
        "work phone",
        "home phone"},
    new String[]{    // custom property values
        "6543212",
        "5059927"},
    "subscribers"); // default domain
```

removeSubscriber

- Syntax, page 3-15
- Description, page 3-16
- Parameters, page 3-16
- Return Value, page 3-16
- RPC Exception Error Codes, page 3-16
- Example, page 3-16

Syntax

```java
public boolean removeSubscriber(String subscriberName)
    throws InterruptedException, OperationTimeoutException, RpcErrorException
```
Description

Removes a subscriber completely from the SM database.

Parameters

subscriberName—See explanation of Subscriber Name Format, page 2-4.

Return Value

- TRUE—If the subscriber was found in the database and successfully removed.
- FALSE—If the conditions for TRUE were not met; i.e., the subscriber was not found in the database, or the subscriber was found but was not successfully removed.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Example

To remove subscriber john entirely from the database:

boolean isExist = removeSubscriber("john");

removeAllSubscribers

- Syntax, page 3-16
- Description, page 3-16
- Return Value, page 3-17
- RPC Exception Error Codes, page 3-17

Syntax

public void removeAllSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException

Description

Removes all subscribers from the SM, leaving the database with no subscribers.
Note
This method may take time to execute. To avoid operation timeout exceptions, set a high operation timeout (up to 5 minutes) before calling this method.

Return Value
None.

RPC Exception Error Codes
None.

getNumberOfSubscribers

- Syntax, page 3-17
- Description, page 3-17
- Return Value, page 3-17
- RPC Exception Error Codes, page 3-17

Syntax

```java
public int getNumberOfSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description
Retrieves the total number of subscribers in the SM database.

Return Value
The number of subscribers in the SM.

RPC Exception Error Codes
None.

getNumberOfSubscribersInDomain

- Syntax, page 3-18
- Description, page 3-18
- Parameters, page 3-18
- Return Value, page 3-18
- RPC Exception Error Codes, page 3-18
Information About Blocking API Methods

Syntax

```java
public int getNumberOfSubscribersInDomain(String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Retrieves the number of subscribers in a subscriber domain.

Parameters

domain—A name of a subscriber domain that exists in the SM's domain repository.

Return Value

The number of subscribers in the domain provided.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DOMAIN_NOT_FOUND

For a description of error codes, see List of Error Codes, page A-1.

getSubscriber

- Syntax, page 3-18
- Description, page 3-18
- Parameters, page 3-19
- Return Value, page 3-19
- RPC Exception Error Codes, page 3-19
- Example, page 3-19

Syntax

```java
public Object[] getSubscriber(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Retrieves a subscriber record. Each field is formatted as an integer, string, or string array, as described below in the Return Value section for this method.

If the subscriber does not exist in the SM database, an exception will be returned.
Parameters

**subscriberName**—See explanation of Subscriber Name Format, page 2-4.

Return Value

An Object Array with nine elements. The following table lists the index values. No array element is NULL.

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>subscriber name (java.lang.String)</td>
</tr>
<tr>
<td>1</td>
<td>array of mappings (java.lang.String[])</td>
</tr>
<tr>
<td>2</td>
<td>array of mapping types (short[])</td>
</tr>
<tr>
<td>3</td>
<td>domain name (java.lang.String)</td>
</tr>
<tr>
<td>4</td>
<td>array of property names (java.lang.String[])</td>
</tr>
<tr>
<td>5</td>
<td>array of property values (java.lang.String[])</td>
</tr>
<tr>
<td>6</td>
<td>array of custom property names (java.lang.String[])</td>
</tr>
<tr>
<td>7</td>
<td>array of custom property values (java.lang.String[])</td>
</tr>
<tr>
<td>8</td>
<td>autologout time, as seconds from now, or -1 if not set (long[])</td>
</tr>
</tbody>
</table>

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Example

To retrieve the subscriber record of *john*:

```java
Object[] subRecord = getSubscriber("john");
String[] mappings = (String[])subRecord[1];
short[] mappingTypes = (short[])subRecord[2];
String domainName = (String)subRecord[3];
String[] propertyNames = (String[])subRecord[4];
String[] propertyValues = (String[])subRecord[5];
String[] customPropertyName = (String[])subRecord[6];
String[] customPropertyValues = (String[])subRecord[7];
long[] autoLogoutTime = (long[])subRecord[8];
```

**subscriberExists**

- Syntax, page 3-20
- Description, page 3-20
### subscriberExists

**Syntax**

```java
public boolean subscriberExists(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

**Description**

Verifies that a subscriber exists in the SM database.

**Parameters**

- `subscriberName`—See explanation of Subscriber Name Format, page 2-4.

**Return Value**

- TRUE—If the subscriber was found in the SM database.
- FALSE—If the subscriber could not be found.

**RPC Exception Error Codes**

None.

### subscriberLoggedIn

**Syntax**

```java
public boolean subscriberLoggedIn(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

**Description**

Checks whether a subscriber that already exists in the SM database is logged in; i.e., if the subscriber also exists in an SCE database.
When the SM is configured to work in Pull mode, a TRUE value returned by this method does not guarantee that the subscriber actually exists in an SCE database, but rather the subscriber is available to be pulled by an SCE if needed. If the subscriber does not exist in the SM database, an exception will be thrown.

**Parameters**

**subscriberName**—See explanation of Subscriber Name Format, page 2-4.

**Return Value**

- TRUE—If the subscriber is logged in.
- FALSE—If the subscriber is not logged in.

**RPC Exception Error Codes**

The following is the list of error codes that this method might return:
- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

**getSubscriberNameByMapping**

- Syntax, page 3-21
- Description, page 3-21
- Parameters, page 3-21
- Return Value, page 3-22
- RPC Exception Error Codes, page 3-22

**Syntax**

```java
public String getSubscriberNameByMapping(String mapping,
                                          short mappingType,
                                          String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

**Description**

Finds a subscriber name according to a mapping and a domain.

**Parameters**

**mapping**—See explanation of mappings and mapping types in the Information About Network ID Mappings section.
**mappingType**—See explanation of mappings and mapping types in the Information About Network ID Mappings section.

**domain**—The name of the domain to which the subscriber belongs. The operation will fail if either of the following conditions exists:

- The domain is null, but the subscriber exists in the database and belongs to a domain.
- The specified domain is incorrect.

**Return Value**

- Subscriber name—If a subscriber record was found.
- NULL—If no subscriber record could be found.

**RPC Exception Error Codes**

The following is the list of error codes that this method might return:

- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

**getSubscriberNames**

- Syntax, page 3-22
- Description, page 3-22
- Parameters, page 3-23
- Return Value, page 3-23
- RPC Exception Error Codes, page 3-23
- Example, page 3-23

**Syntax**

```java
public String[] getSubscriberNames(String lastBulkEnd,
        int numOfSubscribers)
        throws InterruptedException, OperationTimeoutException, RpcErrorException
```

**Description**

Gets a bulk of subscriber names from the SM database, starting with `lastBulkEnd` followed by the next `numOfSubscribers` subscribers (in alphabetical order).

If `lastBulkEnd` is NULL, the (alphabetically) first subscriber name that exists in the SM database will be used.
Chapter 3      Blocking API

Information About Blocking API Methods

Note

There is no guarantee that the total number of subscribers (in all bulks) will equal the value returned from `getNumOfSubscribers` at any time. This may differ, for example, if some subscribers are added or removed while bulks are being retrieved.

Parameters

lastBulkEnd—Last subscriber name from last bulk. Use NULL to start with the first (alphabetic) subscriber.

numOfSubscribers—Limit on number of subscribers that will be returned. If this value is higher than the SM limit (1000), the SM limit will be used.

Note

Providing values higher than 500 to this parameter is not recommended.

Return Value

An array of subscriber names ordered alphabetically.

The method will return as many subscribers as are found in the SM database, starting at the requested subscriber. The array size is limited by the minimum between `numOfSubscribers` and the SM limit (1000).

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Example

```java
boolean hasMoreSubscribers;
String lastBulkEnd = null;
int bulkSize = 100;

do {
    String[] subscribers = smApi.getSubscriberNames(lastBulkEnd, bulkSize);
    hasMoreSubscribers = false;
    if (subscribers != null) {
        for (int i = 0; i < subscribers.length; i++) {
            // do something with subscribers[i]
        }
        if (subscribers.length == bulkSize) {
            hasMoreSubscribers = true;
            lastBulkEnd = subscribers[bulkSize - 1];
        }
    }
} while (hasMoreSubscribers);
```
**getSubscriberNamesInDomain**

- Syntax, page 3-24
- Description, page 3-24
- Parameters, page 3-24
- Return Value, page 3-24
- RPC Exception Error Codes, page 3-24

**Syntax**

```java
public String[] getSubscriberNamesInDomain(String lastBulkEnd,
                                          int numOfSubscribers,
                                          String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

**Description**

Gets subscribers in the SM database that are associated with the specified domain.

The semantics of this operation are the same as the semantics of the getSubscriberNames operation.

**Parameters**

- `lastBulkEnd`—See description in the Parameters section of the getSubscriberNames operation.
- `numOfSubscribers`—See description in the Parameters section of the getSubscriberNames operation.
- `domain`—The name of a subscriber domain that exists in the SM domain repository.

**Return Value**

An alphabetically ordered array of subscriber names that belong to the domain provided.

See also the Return Value section of the getSubscriberNames operation.

**RPC Exception Error Codes**

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

**getSubscriberNamesWithPrefix**

- Syntax, page 3-25
- Description, page 3-25
Syntax

```java
public String[] getSubscriberNamesWithPrefix(String lastBulkEnd,
                                           int numOfSubscribers,
                                           String prefix)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Gets subscribers in the SM database whose name begins with a specified prefix. The semantics of this operation are the same as the semantics of the `getSubscriberNames` operation.

Parameters

- `lastBulkEnd`—See description in the Parameters section of the `getSubscriberNames` operation.
- `numOfSubscribers`—See description in the Parameters section of the `getSubscriberNames` operation.
- `prefix`—A case-sensitive string that marks the prefix of the required subscriber names.

Return Value

An alphabetically ordered array of subscriber names that start with the prefix required. See also the Return Value section of the `getSubscriberNames` operation.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`
- `ERROR_CODE_DATABASE_EXCEPTION`

For a description of error codes, see List of Error Codes, page A-1.

`getSubscriberNamesWithSuffix`

- Syntax, page 3-26
- Description, page 3-26
- Parameters, page 3-26
- Return Value, page 3-26
- RPC Exception Error Codes, page 3-26
Information About Blocking API Methods

Chapter 3      Blocking API

Syntax

public String[] getSubscriberNamesWithSuffix(String lastBulkEnd,
                                             int numOfSubscribers,
                                             String suffix)
throws InterruptedException, OperationTimeoutException, RpcErrorException

Description

Gets subscribers in the SM database whose names end with the specified suffix.
The semantics of this operation are the same as the semantics of the getSubscriberNames operation.

Parameters

lastBulkEnd—See description in the Parameters section of the getSubscriberNames operation.
numOfSubscribers—See description in the Parameters section of the getSubscriberNames operation.
suffix—A case-sensitive string that marks the suffix of the required subscriber names.

Return Value

An alphabetically ordered array of subscriber names that end with the suffix required.
See also the Return Value section of the getSubscriberNames operation.

RPC Exception Error Codes

The following is the list of error codes that this method might return:
• ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
• ERROR_CODE_DATABASE_EXCEPTION
For a description of error codes, see List of Error Codes, page A-1.

getDomains

• Syntax, page 3-26
• Description, page 3-26
• Return Value, page 3-27
• RPC Exception Error Codes, page 3-27

Syntax

public String[] getDomains()
throws InterruptedException, OperationTimeoutException, RpcErrorException

Description

Provides the list of current subscriber domains in the SM domain repository.
Return Value

A complete list of subscriber domain names in the SM.

RPC Exception Error Codes

None.

setPropertiesToDefault

- Syntax, page 3-27
- Description, page 3-27
- Parameters, page 3-27
- Return Value, page 3-27
- RPC Exception Error Codes, page 3-27

Syntax

```java
public void setPropertiesToDefault(String subscriberName,
    String[] properties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Resets the specified application properties of a subscriber. If an application is installed, the relevant application properties will be set to the default value of the properties according to the currently loaded application information. If an application is not installed, a `java.lang.IllegalStateException` will be returned.

Parameters

- `subscriberName`—See explanation of Subscriber Name Format, page 2-4.
- `properties`—See explanation of property keys and values in the Subscriber Properties section.

Return Value

None.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
removeCustomProperties

- Syntax, page 3-28
- Description, page 3-28
- Parameters, page 3-28
- Return Value, page 3-28
- RPC Exception Error Codes, page 3-28

Syntax

```
public void removeCustomProperties(String subscriberName,
    String[] customProperties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Resets the specified custom properties of a subscriber.

Parameters

- `subscriberName`—See explanation of Subscriber Name Format, page 2-4.
- `CustomProperties`—See explanation of custom property keys and values in the Custom Properties section.

Return Value

None.

RPC Exception Error Codes

The following is the list of error codes that this method might return:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes, page A-1.

Blocking API Code Examples

This section contains two code examples:
• Getting number of subscribers
• Adding subscriber, printing subscriber information, removing subscriber

Getting Number of Subscribers

The following example prints to stdout the total number of subscribers in the SM database and the number of subscribers in each subscriber domain.

```java
class PrintInfo {
   public static void main (String args[]) throws Exception {
      SMBlockingApi bapi = new SMBlockingApi();
      try {
         bapi.setReplyTimeout(300000);  //set timeout for 5 minutes
         bapi.connect(args[0]);         // connect to the SM

         String[] domains=bapi.getDomains();
         int totalSubscribers=bapi.getNumberOfSubscribers();
         System.out.println(
            "number of susbcribers in the database:		 " +
            totalSubscribers);
         for (int i=0; i<domains.length; i++) {
            int numberOfSusbcribersInDomain=
               bapi.getNumberOfSubscribersInDomain(domains[i]);
            System.out.println(
               "+number of susbcribers domain "+domains[i]+":		 "+
               numberOfSusbcribersInDomain);
            }
         } finally {
            bapi.disconnect();
         }
   }
}
```

Adding a Subscriber, Printing Information, and Removing a Subscriber

The following program adds a subscriber to the subscriber database, then gets its information and prints it to stdout, and finally removes the subscriber from the subscriber database.

```java
class AddPrintRemove {
   public static void main (String args[]) throws Exception {
      checkArguments(args);
      SMBlockingApi bapi = new SMBlockingApi();
      try {
         bapi.setReplyTimeout(10000);   //set timeout for 10 seconds
         bapi.connect(args[0]);         // connect to the SM
         //add subscriber
```
System.out.println("+ adding subscriber to SM");
bapi.addSubscriber(
    args[1], //name
    new String[]{args[2]}, //mapping
    SMApiConstants.ALL_IP_MAPPINGS,
    new String[]{args[3]}, //property key
    new String[]{args[4]}, //property value
    new String[]{"custom-key"}, //custom property key
    new String[]{"10"}, //custom property value
    args[5]); //domain

//Print subscriber
System.out.println("+ Printing subscriber");
Object[] subfields = bapi.getSubscriber(args[1]);
System.out.println("\tname: \t" + subfields[0]);
System.out.println("\tmapping: \t" + ((String[])subfields[1])[0]);
System.out.println("\tdomain: \t" + subfields[3]);
System.out.println("\tautologout: \t" + subfields[8]);
//Remove subscriber
System.out.println("+ removing subscriber from SM");
bapi.removeSubscriber(args[1]);
}
}

static void checkArguments(String[] args) throws Exception{
    if (args.length != 6) {
        System.err.println(
            "usage: java AddPrintRemove <SM-address> "+
            " <subscriber-name> <IP mapping> <property-key>"+
            " <property-value> <domain> ");
        System.exit(1);
    }
}
Nonblocking API

This module introduces features unique to the Nonblocking API. It lists all methods of the Nonblocking API and ends with code examples.

- Information About Reliability Support, page 4-1
- Autoreconnect Support, page 4-2
- Multithreading Support, page 4-2
- Information About the ResultHandler Interface, page 4-2
- Nonblocking API Construction, page 4-4
- Nonblocking API Initialization, page 4-5
- Information About Nonblocking API Methods, page 4-6
- Nonblocking API Code Examples, page 4-8

Information About Reliability Support

The Nonblocking API can work in two different modes, reliable and nonreliable, as described below. When the mode is not specified, the default is reliable mode.

- Reliable Mode, page 4-1
- Nonreliable Mode, page 4-2

Reliable Mode

In reliable mode, the API ensures that no requests to the SM are lost. The API maintains an internal storage for all API requests that are sent to the SM. After a reply from the SM is received, the request is considered committed and the API can remove the request from its internal storage. In case of connection failure between the API and the SM, the API accumulates all requests in its internal storage until the connection to the SM is established. On reconnection, the API resends all noncommitted requests to the SM, so that no requests are lost.

Note

In reliable mode, the order of resending requests is guaranteed. The API resends the requests in the same chronological order that they were called.
Nonreliable Mode

In nonreliable mode, the API does not ensure that requests sent to the SM are executed. In addition, all requests that are sent by the API when the connection to the SM is down will be lost unless an external reliability mechanism is implemented.

Autoreconnect Support

The Nonblocking API supports autoreconnection to the SM in case of connection failure. When this option is activated, the API can determine when the connection to the SM is lost. When the connection is lost, the API activates a reconnection task that tries to reconnect to the SM until it is successful.

Note

The autoreconnect support option can be activated regardless of the reliability mode.

Multithreading Support

The Nonblocking API supports an unlimited number of threads calling its methods simultaneously.

Note

In a multithreaded scenario for the Nonblocking API, the order of invocation is guaranteed. The API performs operations in the same chronological order that they were called.

Information About the ResultHandler Interface

The Nonblocking API enables setting a result handler. A result handler is an interface with two methods, handleSuccess and handleError, as outlined in the following code:

```java
public interface ResultHandler {
    /**
     * handle a successful result
     */
    public void handleSuccess(long handle, Object result);

    /**
     * handle a failure result
     */
    public void handleError(long handle, Object result);
}
```

You should implement this interface if you want to be informed about the success/error results of operations performed through the API.

Note

This is the only interface for retrieving results; they cannot be returned immediately after the API method has returned to the caller.
In order to be able to receive operation results, you should set the result handler of the API before calling API methods whose results you want to receive. It is a good practice to set the result handler after the API is connected (as in the example below).

Both **handleSuccess** and **handleError** methods accept two parameters:

- **Handle**—Each API operation's return-value is a handle of type `long`. This handle enables correlation between operation calls and their results. When a **handle**... operation is called with a handle of value `X`, the result will match the operation that returned the same handle value (`X`) to the caller.
- **Result**—The actual result of the operation. Some operations may return a result of `NULL`.

### ResultHandler Interface Example

The following example is a simple implementation of a result handler that prints a message to `stdout` (when the result is successful) or to `stderr` (when the result is failure). This main method initiates the API and assigns a result handler.

For correct operation of the result handler, follow the code sequence given in this example.

**Note** This example does not demonstrate the use of callback handles.

```java
import com.pcube.management.framework.rpc.ResultHandler;
import com.pcube.management.api.SMNonBlockingApi;

public class ResultHandlerExample implements ResultHandler{
    public void handleSuccess(long handle, Object result) {
        System.out.println("success: handle="+handle+", result="+result);
    }

    public void handleError(long handle, Object result) {
        System.err.println("error: handle="+handle+", result="+result);
    }

    public static void main (String args[]) throws Exception{
        if (args.length != 1) {
            System.err.println("usage: ResultHandlerExample <sm-ip>");
            System.exit(1);
        }

        //note the order of operations!
        SMNonBlockingApi nbapi = new SMNonBlockingApi();
        nbapi.connect(args[0]);
        nbapi.setResultHandler(new ResultHandlerExample());
        nbapi.login(...);
    }
}
```
Nonblocking API Construction

In addition to the constructors described in API Construction, page 2-2, the Nonblocking API provides constructors that enable setting the reconnect period and the reliability mode.

Nonblocking API Syntax

The syntax for the additional Nonblocking API constructors is shown in the following code block:

```java
public SMNonBlockingApi(long autoReconnectInterval)
public SMNonBlockingApi(boolean reliable, long autoReconnectInterval)
public SMNonBlockingApi(String legName, long autoReconnectInterval)
public SMNonBlockingApi(String legName, boolean reliable, long autoReconnectInterval)
```

Nonblocking API Arguments

The following is a description of the constructor arguments for the additional Nonblocking API constructors:

- **autoReconnectInterval**
  
  Defines the interval (in milliseconds) for attempting reconnection by the reconnection task, as follows:
  
  - If the value is 0 or less, the reconnection task is not activated (no autoreconnect is attempted).
  - If the value is greater than 0 and if there is a connection failure, the reconnection task will be activated every `autoReconnectInterval` milliseconds.
  
  The default value is -1 (no autoreconnect is attempted).

- **reliable**
  
  A flag that defines whether the API should work in reliable mode, as follows:
  
  - TRUE—The API works in reliable mode.
  - FALSE—The API works in nonreliable mode.

  The default value is TRUE (the API works in reliable mode).

- **legName**
  The name of the LEG, as described in API Construction, page 2-2.

Note: To enable the autoreconnect support, the connect method of the API must be activated at least once. For more information see, Nonblocking API Code Examples, page 4-8.

Nonblocking API Examples

The following code constructs a reliable API with an autoreconnection interval of 10 seconds:

```java
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);
nbapi.connect(<SM IP address>);
```

The following code constructs a reliable API without autoreconnection support:

```java
SMNonBlockingAPI nbapi = SMNonBlockingAPI();
nbapi.connect(<SM IP address>);
```
Nonblocking API Initialization

The Nonblocking API enables initializing certain internal properties for API customization. This initialization is performed using the API `init` method.

**Note**

For the settings to take effect, the `init` method must be called **before** the `connect` method.

The following properties can be set:

- **Output queue size**—The internal buffer size defining the maximum number of requests that can be accumulated by the API until they are sent to the SM. The default is 1024.

- **Operation timeout**—The desired timeout (in milliseconds) on a nonresponding PRPC protocol connection. The default is 45 seconds.

Nonblocking API Initialization Syntax

The syntax for the Nonblocking API `init` method is as follows:

```java
public void init(Properties properties)
```

Nonblocking API Initialization Parameters

The following is a description of the parameters for the Nonblocking API `init` method:

- **properties** (`java.util.Properties`)
  
  Enables setting the following properties described above:

  - To set the output queue size, use `prpc.client.output.machinemode.recordnum`
  
  - To set the operation timeout, use `prpc.client.operation.timeout`

Nonblocking API Initialization Example

The following code illustrates how to customize properties during initialization when using the Nonblocking API. Note that the `init` method is called **before** the `connect` method.

```java
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);
// API initialization
java.util.Properties p = new java.util.Properties();
p.setProperty("prpc.client.output.machinemode.recordnum", 2048);
```
p.setProperty("prpc.client.operation.timeout", 60000);// 1 minute
nbapi.init(p);
// initial connect to the API to enable the reconnect task
nbapi.connect(<SM API address>);

Information About Nonblocking API Methods

This section describes the methods of the Nonblocking API.
All methods return a handle of type long that can be used to correlate operation calls and their results.
See Information About the ResultHandler Interface, page 4-2.
The operation results passed to the result handler are similar to the return values described in the same
method in the Blocking API, with the exception of:
- Basic types are converted to their Java class representation. For example, int is translated to
  java.lang.Integer.
- Return values of Void are translated to NULL.

Note
An error will be passed to the result handler only if the matching operation in the Blocking API throws
an exception with the same arguments according to the SM database state at the time of the call.

All methods will throw a java.lang.IllegalArgumentException if called before a connection with the SM is
established.

This section describes the following methods:
- login, page 4-6
- logoutByName, page 4-7
- logoutByNameFromDomain, page 4-7
- logoutByMapping, page 4-7
- loginCable, page 4-7
- logoutCable, page 4-8

login

Syntax

public long login(String subscriberName,
                   String[] mappings,
                   short[] mappingTypes,
                   String[] propertyKeys,
                   String[] propertyValues,
                   String domain,
                   boolean isMappingAdditive,
                   int autoLogoutTime)

The operation functionality is the same as the matching Blocking API operation. See login for more
information.
**logoutByName**

**Syntax**

```java
public long logoutByName(String subscriberName,
    String[] mappings,
    short[] mappingTypes)
```

The operation functionality is the same as the matching Blocking API operation. See `logoutByName` for more information.

**logoutByNameFromDomain**

**Syntax**

```java
public long logoutByNameFromDomain(String subscriberName,
    String[] mappings,
    short[] mappingTypes,
    String domain)
```

The operation functionality is the same as the matching Blocking API operation. See `logoutByNameFromDomain` for more information.

**logoutByMapping**

**Syntax**

```java
public long logoutByMapping(String mapping,
    short mappingType,
    String domain)
```

The operation functionality is the same as the matching Blocking API operation. See `logoutByMapping` for more information.

**loginCable**

**Syntax**

```java
public long loginCable(String CPE,
    String CM,
    String IP,
    int lease,
    String domain,
    String[] propertyKeys,
    String[] propertyValues)
```

The operation functionality is the same as the matching Blocking API operation. See `loginCable` for more information.
logoutCable

Syntax

    public long logoutCable(String CPE,
                            String CM,
                            String IP,
                            String domain)

The operation functionality is the same as the matching Blocking API operation. See logoutCable for more information.

Nonblocking API Code Examples

This section illustrates a code example for logging in and logging out subscribers.

Login and Logout

The following example logs in a predefined number of subscribers to the SM and then logs them out. Note the implementation of a disconnect listener and a result handler.

    package nonblocking;
    import com.pcube.management.framework.rpc.DisconnectListener;
    import com.pcube.management.framework.rpc.ResultHandler;
    import com.pcube.management.api.SMNonBlockingApi;
    import com.pcube.management.api.SMApiConstants;

    class LoginLogoutDisconnectListener implements DisconnectListener {
        public void connectionIsDown() {
            System.err.println("disconnect listener:: connection is down");
        }
    }

    class LoginLogoutResultHandler implements ResultHandler {
        int count = 0;

        //prints a success result every 100 results
        public synchronized void handleSuccess(long handle, Object result) {
            Object tmp = null;
            if (++count%100 == 0) {
                tmp = result instanceof Object[] ? ((Object[])result)[0] : result;
                System.out.println("result "+count+":	"+tmp);
            }
        }

        //prints every error that occurs
        public synchronized void handleError(long handle, Object result) {
            System.err.println("error: "+count+":	"+result);
            ++count;
        }

        //waits for result number 'last result' to arrive
        public synchronized void waitForLastResult(int lastResult) {
            while (count<lastResult) {
                try {
                    wait(100);
                }
            }
        }
    }
public class LoginLogout {
    public static void main (String args[]) throws Exception{
        //check arguments
        checkArguments(args);
        int numSubscribersToLogin = Integer.parseInt(args[2]);

        //instantiation
        SMNonBlockingApi nbapi = new SMNonBlockingApi();
        try {
            //initiation
            nbapi.setDisconnectListener(new LoginLogoutDisconnectListener);
            nbapi.connect(args[0]);
            LoginLogoutResultHandler resultHandler = new LoginLogoutResultHandler();
            nbapi.setResultHandler(resultHandler);

            //login
            System.out.println("login of "+numSubscribersToLogin +" subscribers");
            for (int i=0; i<numSubscribersToLogin; i++) {
                nbapi.login("subscriber"+i, //subscriber name
                    getMappings(i), //a single ip mapping
                    new short[]{SMApiConstants.MAPPING_TYPE_IP, null, //no properties
                        null, //domain
                        false, //mappings are not additive
                        -1}); //disable autologout
            }
            resultHandler.waitForLastResult(numSubscribersToLogin);

            //logout
            System.out.println("logout of "+numSubscribersToLogin +" subscribers");
            for (int i=0; i<numSubscribersToLogin; i++) {
                nbapi.logoutByMapping(getMappings(i)[0], SMApiConstants.MAPPING_TYPE_IP, args[1]);
            }
            resultHandler.waitForLastResult(numSubscribersToLogin*2);
        } finally {
            nbapi.disconnect();
        }
    }
}

static void checkArguments(String[] args) throws Exception{
    if (args.length != 3) {
        System.err.println("usage: java LoginLogout "+
            "<SM-address> <domain> <num-subscribers>");
        System.exit(1);
    }
}

private static String[] getMappings(int i) {
    return new String[]{ "10." +((int)i/65536)%256 + "." + ((int)(i/256))%256 + "." + (i%256)};
List of Error Codes

This module provides a list of error codes that are used in the Java API.

List of Error Codes

Error codes are used for interpreting the actual error for which an RpcErrorException was returned. Use the getErrorCode method to extract the error code.

The error code enumeration is given in the com.pcube.management.api.SMApiConstants interface. The following table gives a list of the error codes and their descriptions.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CODE_ARRAY_ACCESS</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_ATTRIBUTE_NOT_FOUND</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_BAD_SUBSCRIBER_MAPPING</td>
<td>A mapping was formatted badly or assigned to the subscriber illegally.</td>
</tr>
<tr>
<td>ERROR_CODE_CLASS_CAST</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_CLASS_NOT_FOUND</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_CLIENT_INTERNAL_ERROR</td>
<td>Internal error.</td>
</tr>
<tr>
<td>ERROR_CODE_CLIENT_OUT_OF_THREADS</td>
<td>Internal error.</td>
</tr>
<tr>
<td>ERROR_CODE_DATABASE_EXCEPTION</td>
<td>Internal SM error – database error occurred during the operation.</td>
</tr>
<tr>
<td>ERROR_CODE_DOMAIN_NOT_FOUND</td>
<td>The domain provided to the operation does not exist in the SM domain repository.</td>
</tr>
<tr>
<td>ERROR_CODE_ILLEGAL_ARGUMENT</td>
<td>One of the arguments provided to the method is illegal.</td>
</tr>
<tr>
<td>ERROR_CODE_ILLEGAL_STATE</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME</td>
<td>The subscriber name provided has more than 40 characters or has illegal characters.</td>
</tr>
<tr>
<td>ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN</td>
<td>The domain provided to the operation exists in the SM domain repository but is not a subscriber domain.</td>
</tr>
</tbody>
</table>
### Table A-1 List of Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CODE_NULL_POINTER</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_NUMBER_FORMAT</td>
<td>A VLAN mapping string provided to the API does not represent a decimal number.</td>
</tr>
<tr>
<td>ERROR_CODE_OBJECT_NOT_FOUND</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_OPERATION_NOT_FOUND</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_OUT_OF_MEMORY</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_RUNTIME</td>
<td>Internal SM error.</td>
</tr>
<tr>
<td>ERROR_CODE_SE_ERROR</td>
<td>Internal SM error. The SM could not perform the operation on the SCE device.</td>
</tr>
<tr>
<td>ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST</td>
<td>The subscriber on which the operation is performed does not exist in the SM database.</td>
</tr>
<tr>
<td>ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION</td>
<td>The subscriber exists in the SM database but is associated with a domain other than the one specified by the operation.</td>
</tr>
<tr>
<td>ERROR_CODE_SUBSCRIBER_MAPPING_CONGESTION</td>
<td>The mappings provided for the subscriber by the operation already belong to another subscriber.</td>
</tr>
<tr>
<td>ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS</td>
<td>The subscriber on which the operation was performed already exists in the SM database.</td>
</tr>
<tr>
<td>ERROR_CODE_UNKNOWN</td>
<td>Internal SM or API error.</td>
</tr>
</tbody>
</table>