



SCMS SM Java API Programmer's Guide

OL-7204-01

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number: DOC-138634=
Text Part Number: OL-7204-01



THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The following information is for FCC compliance of Class A devices: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio-frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case users will be required to correct the interference at their own expense.

The following information is for FCC compliance of Class B devices: The equipment described in this manual generates and may radiate radio-frequency energy. If it is not installed in accordance with Cisco's installation instructions, it may cause interference with radio and television reception. This equipment has been tested and found to comply with the limits for a Class B digital device in accordance with the specifications in part 15 of the FCC rules. These specifications are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation.

Modifying the equipment without Cisco's written authorization may result in the equipment no longer complying with FCC requirements for Class A or Class B digital devices. In that event, your right to use the equipment may be limited by FCC regulations, and you may be required to correct any interference to radio or television communications at your own expense.

You can determine whether your equipment is causing interference by turning it off. If the interference stops, it was probably caused by the Cisco equipment or one of its peripheral devices. If the equipment causes interference to radio or television reception, try to correct the interference by using one or more of the following measures:

- Turn the television or radio antenna until the interference stops.
- Move the equipment to one side or the other of the television or radio.
- Move the equipment farther away from the television or radio.
- Plug the equipment into an outlet that is on a different circuit from the television or radio. (That is, make certain the equipment and the television or radio are on circuits controlled by different circuit breakers or fuses.)

Modifications to this product not authorized by Cisco Systems, Inc. could void the FCC approval and negate your authority to operate the product.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCSP, the Cisco Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Access Registrar, Aironet, ASIST, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Empowering the Internet Generation, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, IQ Expertise, the IQ logo, IQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, Packet, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, ScriptShare, SlideCast, SMARTnet, StrataView Plus, SwitchProbe, TeleRouter, The Fastest Way to Increase Your Internet Quotient, TransPath, and VCO are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0501R)

Printed in the USA on recycled paper containing 10% postconsumer waste.

SCMS SM Java API Programmer's Guide

Copyright © 2002-2005 Cisco Systems, Inc.
All rights reserved.



Preface v

Audience v

Purpose v

Related Publications v

Document Conventions vi

Technical Support vi

 Cisco TAC Website vi

 Opening a TAC Case vii

 TAC Case Priority Definitions vii

Getting Started 1-1

Introduction 1-1

Platforms 1-1

Installation 1-2

 Extracting the Package 1-2

Compiling and running 1-3

SM setup 1-3

General API Concepts 2-1

Blocking API/Non-blocking API 2-1

 Blocking API 2-1

 Non-blocking API 2-2

API Initialization 2-2

 API Construction 2-2

 Setup Operations 2-3

 Connecting to the SM 2-3

API Finalization 2-4

Subscriber Name Format 2-4

Network ID Mappings 2-4

- Specifying IP Address Mapping 2-5
- Specifying IP Range Mapping 2-5
- Specifying VLAN Tag Mapping 2-5
- Subscriber Domains 2-6
- Subscriber Properties 2-6
- Custom Properties 2-6
- DisconnectListener Interface 2-7
- Exceptions 2-7

Blocking API 3-1

- Multi-threading Support 3-1
- ReplyTimeout and OperationTimeout Exception 3-2
- Blocking API Methods 3-3
 - login 3-4
 - logoutByName 3-7
 - logoutByNameFromDomain 3-8
 - logoutByMapping 3-9
 - loginCable 3-10
 - logoutCable 3-12
 - addSubscriber 3-13
 - removeSubscriber 3-15
 - removeAllSubscribers 3-16
 - getNumberOfSubscribers 3-16
 - getNumberOfSubscribersInDomain 3-16
 - getSubscriber 3-17
 - subscriberExists 3-18
 - subscriberLoggedIn 3-19
 - getSubscriberNameByMapping 3-19
 - getSubscriberNames 3-20
 - getSubscriberNamesInDomain 3-22
 - getSubscriberNamesWithPrefix 3-22
 - getSubscriberNamesWithSuffix 3-23
 - getDomains 3-24
 - setPropertiesToDefault 3-25

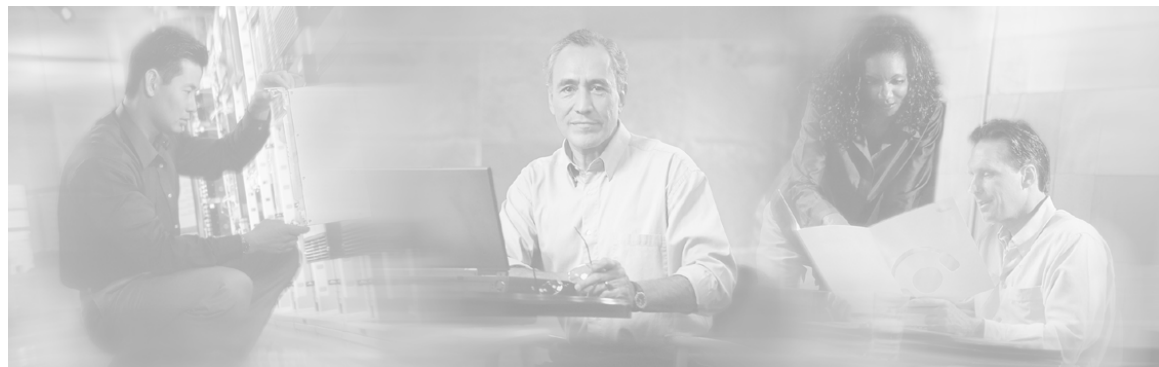
- removeCustomProperties 3-25
- Blocking API Code Examples 3-26
 - Getting Number of Subscribers 3-26
 - Adding Subscriber, Printing Information, Removing Subscriber 3-27

Non-blocking API 4-1

- Reliability Support 4-1
 - Reliable Mode 4-2
 - Non-reliable Mode 4-2
- Auto-reconnect Support 4-2
- Multi-threading Support 4-2
- ResultHandler Interface 4-3
- Non-blocking API Construction 4-4
- Non-blocking API Initialization 4-5
- Non-blocking API Methods 4-6
 - login 4-7
 - logoutByName 4-7
 - logoutByNameFromDomain 4-8
 - logoutByMapping 4-8
 - loginCable 4-8
 - logoutCable 4-8
- Non-blocking API Code Examples 4-8
 - Login and Logout 4-9

List of Error Codes A-1

Index 1



Preface

The *SCMS SM Java API* is used for updating, querying, and configuring the SM. It consists of two parts, which may be used separately or together without limitation.

- **SM Non-blocking Java API:** A high-performance API with low visibility to errors and other operation results. Supports automatic integrations with OSS/AAA systems.
- **SM Blocking Java API:** A more user-friendly API. Supports user interface applications for accessing and managing the SM.



Note

A set of APIs with exactly the same functionality is available in C and C++ as well.

Audience

This guide is for the networking or computer technician responsible for configuring the SM. It is also intended for the operator who manages the SCE Platform(s).

Purpose





This document explains the *SCMS SM Java API*, and explains how to install, compile, and run it.

Related Publications

This API Guide should be used in conjunction with the SCMS Subscriber Manager suite of User, API and Reference Guides.

Document Conventions

The following typographic conventions are used in this guide:

Typeface or Symbol	Meaning
<i>Italics</i>	References, new terms, field names, and placeholders.
Bold	Names of menus, options, and command buttons.
Courier	System output shown on the computer screen in the Telnet session.
Courier Bold	CLI code typed in by the user in examples.
<i>Courier Italic</i>	Required parameters for code.
[<i>italic in brackets</i>]	Optional parameters for code.
	Note.
	Notes contain important information.
	Warning.
	Warning means danger of bodily injury or of damage to equipment.

Technical Support

Cisco TAC Website

The Cisco TAC website (<http://www.cisco.com/tac> (<http://www.cisco.com/tac>)) provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The Cisco TAC website is available 24 hours a day, 365 days a year.

Accessing all the tools on the Cisco TAC website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a login ID or password, register at this URL: <http://tools.cisco.com/RPF/register/register.do> (<http://tools.cisco.com/RPF/register/register.do>)

Opening a TAC Case

The online TAC Case Open Tool (<http://www.cisco.com/tac/caseopen>) is the fastest way to open P3 and P4 cases. (Your network is minimally impaired or you require product information). After you describe your situation, the TAC Case Open Tool automatically recommends resources for an immediate solution.

If your issue is not resolved using these recommendations, your case will be assigned to a Cisco TAC engineer. For P1 or P2 cases (your production network is down or severely degraded) or if you do not have Internet access, contact Cisco TAC by telephone. Cisco TAC engineers are assigned immediately to P1 and P2 cases to help keep your business operations running smoothly.

To open a case by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)

EMEA: +32 2 704 55 55

USA: 1 800 553-2447

For a complete listing of Cisco TAC contacts, go to this URL:

<http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml>
(<http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml>)

TAC Case Priority Definitions

To ensure that all cases are reported in a standard format, Cisco has established case priority definitions.

Priority 1 (P1)—Your network is “down” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Priority 2 (P2)—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Priority 3 (P3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Priority 4 (P4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.



Getting Started

This section discusses the platforms on which the Java API can be used, and how to install, compile, and start running it.

This chapter contains the following sections:

- [Introduction](#) 1-1
- [Platforms](#) 1-1
- [Installation](#) 1-2
- [Compiling and running](#) 1-3
- [SM setup](#) 1-3

Introduction

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

- **SM Non-blocking Java API:** A high-performance API with low visibility to errors and other operation results. Supports automatic integrations with OSS/AAA systems.
- **SM Blocking Java API:** A more user-friendly API. 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 1.3 or later.

Installation

Extracting the Package

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

To install the distribution on a UNIX platform, use the following command line:

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

To install the distribution on a Windows platform, use a zip extractor (such as WinZip). The abbreviations *vvv* and *bb* stand for the Java SM API version and build number.

Package Content

For brevity, the installation directory `sm-java-api-vvv.bb` is referred to as `<installdir>`.

The `<installdir>/javadoc` folder contains the API JAVADOC documentation.

The `<installdir>/lib` folder contains the `smapi.jar` file, which is the API Executable. It also contains additional jar files necessary for the API operation.

Table 1-1 Layout of Installation Directory

Path	Name	Description
<installdir>	README	API readme file
	<installdir>/Javadoc	
	Index.html (API specification files, etc.)	Index of all API specifications API specification documents
<installdir>/Lib	smapi.jar	SM API executable
	Asn1rt.jar	Utility jar used by the API
	jdmkrt.jar	Utility jar used by the API
	Log4j.jar	Utility jar used by the API
	Log4j.properties	
	xerces.jar	Utility jar used by the API

Compiling and running

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

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

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

Afterward, use the following command line to run the program:

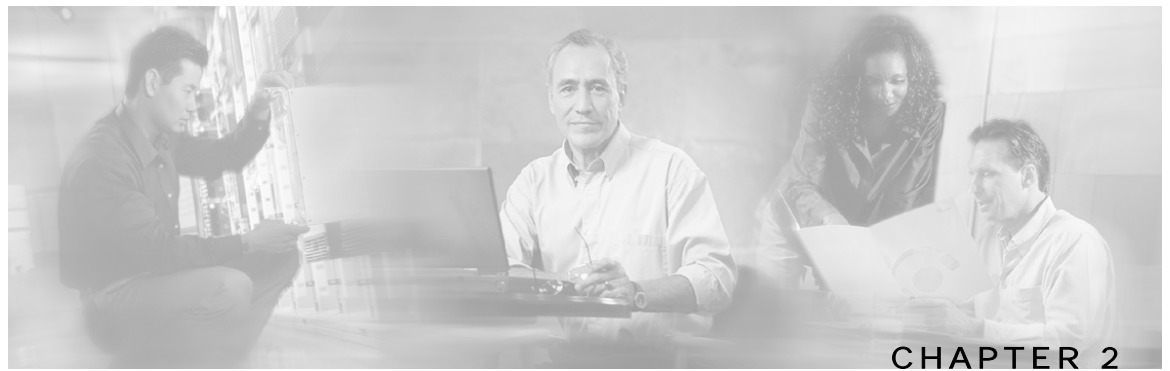
```
#> java -cp .:<installdir>/lib/smapi.jar SMApiProgram
```

SM setup

The API connects to the PRPC server on the SM. In order 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, see the *SCMS Subscriber Manager User Guide*.



General API Concepts

This section describes various concepts that are utilized when working with the SM Java API.

This chapter contains the following sections:

- [Blocking API/Non-blocking API](#) 2-1
- [API Initialization](#) 2-2
- [API Finalization](#) 2-4
- [Subscriber Name Format](#) 2-4
- [Network ID Mappings](#) 2-4
- [Subscriber Domains](#) 2-6
- [Subscriber Properties](#) 2-6
- [Custom Properties](#) 2-6
- [DisconnectListener Interface](#) 2-7
- [Exceptions](#) 2-7

Blocking API/Non-blocking API

Describes the differences between Blocking APIs and Non-blocking APIs.

Blocking API

In a Blocking API, which is the common type, 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 Non-blocking API (reliability and auto-reconnect are not supported), as well as many functions that are not provided by the Non-blocking API.

Non-blocking API

Non-blocking methods return immediately, even before their operation has been completed. The operation results are either returned to an Observer object (Listener) or not returned at all.

The Non-blocking method is advantageous when the operation is lengthy and involves I/O. Performing the operation in a separate thread allows the caller to continue doing other tasks and improves overall system performance.

The SM Non-blocking Java API contains a small number of non-blocking operations. The API supports retrieval of operation results using a result listener.

The SM Non-blocking Java API supports two modes: *reliable* and *non-reliable*. For more information about the reliability modes, see *Reliability Support* (on page 4-1).

API Initialization

To initialize the API:

-
- Step 1** Construct the API using one of its constructors.
 - Step 2** Perform the API-specific setup operations.
 - Step 3** Connect the API to the SM.
-

The three steps above are described in the following sections.

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

API Construction

Blocking and Non-blocking 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 LEGs and SM-LEG failure handling in the *SCMS Subscriber Manager User Guide*.

The LEG name will be used by the SM 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 Non-blocking 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 *SCMS Subscriber Manager User Guide, Appendix A (Configuration File Options)*.

Additional constructors are available for the Non-blocking API. For more information, see *Non-blocking API Construction (on page 4-4)*.

Setup Operations

The setup operations differ for the two APIs. Both APIs support setting a disconnect listener, described in more detail in the *DisconnectListener Interface (on page 2-7)* section.

Blocking API setup

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

Non-blocking API setup

To set up the Non-blocking API you are required to set a disconnect listener. For more details, see the *Non-blocking API (on page 4-1)* chapter.

Connecting to the SM

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

- The following method uses the default RPC TCP port (14374) to connect to the SM.

```
connect(String host)
```

- The following method allows 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 method `isConnected`.

API Finalization

To free the resources of both server and client, call 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.

The subscriber name is *case-sensitive*. It may contain up to 40 characters. The following characters may be used:

Alphanumerics	\$ (dollar sign)	. (period or dot)	_ (underscore)
- (minus sign or hyphen)	% (percent sign)	/ (slash)	~ (tilde)
! (exclamation mark)	& (ampersand)	: (colon)	' (apostrophe)
# (number sign)	() (parentheses)	@ (at sign)	

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. For additional information, see the *SCMS Subscriber Manager User Guide*. Currently, the Cisco Service Control system supports IP address, IP range, and VLAN mappings.

Both Blocking and Non-blocking APIs contain operations that accept mappings as a parameter. Examples are:

- the `addSubscriber` operation (Blocking API)
- the `login` method (Blocking or Non-blocking 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

- VLAN tags

Specifying IP Address Mapping

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

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

Example:

- `216.109.118.66`
- The mapping type of an IP address is provided in the 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 VLAN Tag Mapping

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

The string is simply a decimal in the specified range.

The mapping type is also provided in interface

`com.pcube.management.api.SMApiConstants`:

- `com.pcube.management.api.SMApiConstants.MAPPING_TYPE_VLAN` 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_VLAN_MAPPINGS` specifies that all the entries in the mapping identifiers array are VLAN mappings.

Subscriber Domains

The domain concept is explained in detail in the *SCMS Subscriber Manager User Guide*. Roughly, a domain is an identifier that tells the SM which SCE devices should be updated 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.

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 *SCMS Subscriber Manager User Guide* and in your application's User Guide (*SCAS BB* or *SCAS M*). 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 `{"name", "color", "shape"}` and the property values array is `{"john", "red", "circle"}`, the properties will be `name=john, color=red, shape=circle`.

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* (on page 2-6).

DisconnectListener Interface

Both APIs (Blocking and Non-blocking) 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();

}
```

An API user who wants to be notified when the API is disconnected from the SM should implement this interface.

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

Example:

- Following 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

Note that all functional errors of the SM Java API are provided by the same Java class, `com.pcube.management.framework.rpc.RpcErrorException`, which is *contrary* to normal Java usage. This “contrary” approach was chosen because of the “cross-language” nature of the SM API: it allows all the SM API implementations (Java, C, C++) to look and feel the same.

Each exception provides the following information:

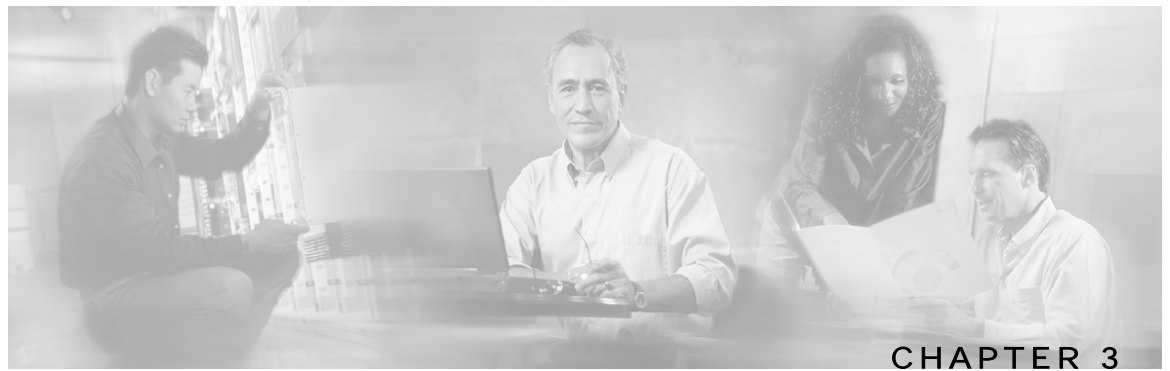
- A unique error code (`long`)
- An informative message (`java.lang.String`)
- A server-side stack trace (`java.lang.String`)

The error code can be interpreted using `com.pcube.management.api.SMApiConstants`. See the *List of Error Codes* (on page [A-1](#)) 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* (on page [3-1](#)) chapter.



Blocking API

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



Note

If you only need to develop an *automatic integration*, skip this chapter and go directly to the *Non-blocking API* (on page 4-1) chapter.

This chapter contains the following sections:

- [Multi-threading Support](#) 3-1
- [ReplyTimeout and OperationTimeout Exception](#) 3-2
- [Blocking API Methods](#) 3-3
- [Blocking API Code Examples](#) 3-26

Multi-threading Support

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

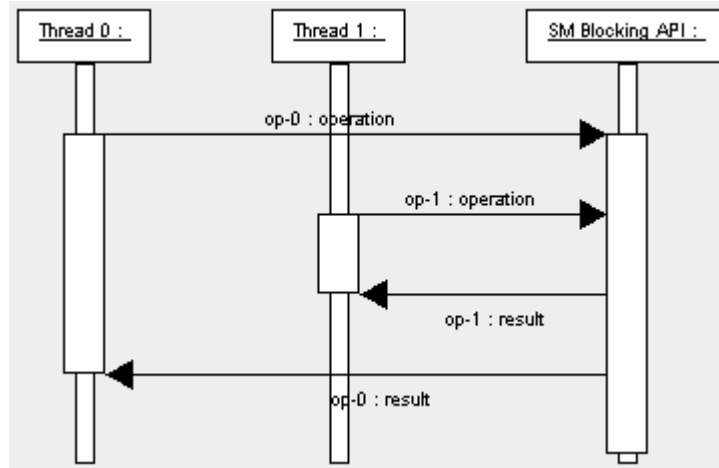


Note

In a multi-threaded 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):



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 alternate way of releasing 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.

Blocking API Methods

This section lists the methods of the Blocking API. The signature of each method is followed by a description of its input parameters and its return values.

The Blocking API is a superset of the Non-blocking API. Except for differences in return values and result handling, identical operations in both APIs have the same semantics.

All the methods throw a `java.lang.IllegalStateException` when called before a connection with the SM is established.

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

- **Dynamic IP and property allocation-** For example, by using the SM API for integration with an AAA system, the following methods are relevant. (Note, 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* (on page [3-4](#))
 - *logoutByName*
 - *logoutByNameFromDomain*
 - *logoutByMapping*
 - *loginCable* (on page [3-10](#))
 - *logoutCable* (on page [3-12](#))
- **Static/Manual Subscriber configuration-** For example, for GUI usage, the following methods are relevant:
 - *addSubscriber* (on page [3-13](#))
 - *removeSubscriber* (on page [3-15](#))
 - *removeAllSubscribers* (on page [3-16](#))
 - *setPropertyToDefault* (on page [3-25](#))
 - *removeCustomProperties* (on page [3-25](#))
- For simple read-only operations, performed independently on the subscriber awareness mode, the following methods are relevant:
 - *getNumberOfSubscribers* (on page [3-16](#))
 - *getNumberOfSubscribersInDomain* (on page [3-16](#))
 - *getSubscriber* (on page [3-17](#))
 - *subscriberExists* ("[RPC Exception Error Codes](#)" on page [3-18](#))
 - *subscriberLoggedIn* (on page [3-19](#))
 - *getSubscriberNameByMapping* (on page [3-19](#))
 - *getSubscriberNames* (on page [3-20](#))
 - *getSubscriberNamesInDomain* (on page [3-22](#))
 - *getSubscriberNamesWithPrefix* (on page [3-22](#))

- *getSubscriberNamesWithSuffix* (on page 3-23)
- *getDomains* (on page 3-24)

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

login

Syntax

```
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 who already exists in the SM database. Login 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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

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* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`propertyKeys`: See explanation of property keys and values in the *General API Concepts* (on page 2-1) chapter.

`propertyValues`: See explanation of property keys and values in the *General API Concepts* (on page 2-1) chapter.

`domain`: See explanation of domains in the *General API Concepts* (on page 2-1) chapter.

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

`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

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`
- `ERROR_CODE_BAD_SUBSCRIBER_MAPPING`
- `ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION`
- `ERROR_CODE_UNKNOWN` can be caused by the following:
 - `NULL` value for `domain` parameter for the subscriber that does not exist/does not have a domain
 - Invalid values for `propertyValues` parameter

For error codes list see Appendix A - List of Error Codes.

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",
    new String[]{"192.168.12.5"},
    SMApiConstants.ALL_IP_MAPPINGS,
    null, null,
    "subscribers",
    true,
    -1);
```

To add the IP address 192.168.12.5 overriding previous mappings:

```
login(
    "john",
    new String[]{"192.168.12.5"},
    SMApiConstants.ALL_IP_MAPPINGS,
    null, null,
    "subscribers",
    false,
    -1);
```

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

```
login(
    "john",
    new String[]{"192.168.12.5"},
    SMApiConstants.ALL_IP_MAPPINGS,
    null, null,
    "subscribers",
    false,
    300);
```

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

```
login(
    "john",
    null, null,
    new String[]{"packageId"},
    new String[]{"10"},
    "subscribers",
    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"},
    new String[]{"10"},
    "subscribers",
    true,
    -1);
```

logoutByName

Syntax

```
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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

If no mappings are specified, all the subscriber mappings will be removed.

`mappingTypes`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

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

Following is the list of error codes that may be returned by this method:

- 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

For error codes description see Appendix A - List of Error Codes.

Example

To remove IP address 192.168.12.5 of subscriber *john*:

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

To remove all IP addresses of subscriber *john*:

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

logoutByNameFromDomain

Syntax

```
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* (on page 2-4) in the *General API Concepts (on page 2-1)* chapter.

`mappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts (on page 2-1)* chapter.

If no mappings are specified, all the subscriber mappings will be removed.

`mappingTypes`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts (on page 2-1)* chapter.

`domain`: See explanation of domains in the *General API Concepts (on page 2-1)* chapter. 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

Following is the list of error codes that may be returned by this method:

- 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

For error codes description see Appendix A - List of Error Codes.

Example

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

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

To remove all IP addresses of subscriber *john* from domain *subscribers*:

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

logoutByMapping

Syntax

```
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* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

mappingType: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

domain: See description in `logoutByNameFromDomain` (on page 3-8) 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

Following is the list of error codes that may be returned by this method:

- 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

For error codes description see Appendix A - List of Error Codes.

Example

To remove IP address 192.168.12.5 from domain *subscribers*:

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

loginCable

Syntax

```
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 *Cable Environment Appendix* of the *SCMS 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 *domains* ("[Subscriber Domains](#)" on page 2-6) in the *General API Concepts* (on page 2-1) chapter.

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 read *Configuring Domains* section of *SCMS Subscriber Manager User Guide*.

propertyKeys: See explanation of *property keys* ("[Subscriber Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

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* ("[Subscriber Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

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

```
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 `loginCable` (on page 3-10) method.

CM: See description in the `loginCable` (on page 3-10) method.

IP: See description in the `loginCable` (on page 3-10) method.

domain: See description in the `loginCable` (on page 3-10) 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*:

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

addSubscriber

Syntax

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

Description

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 auto-logout feature is always disabled. To enable auto-logout, 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 be left with its previous IP mapping: *IP1*.

Parameters

`subscriberName`: See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappingTypes`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`propertyKeys`: See explanation of *property keys* ("[Subscriber Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`propertyValues`: See explanation of *property keys* ("[Subscriber Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`customPropertyKeys`: See explanation of *custom property keys* ("[Custom Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`customPropertyValues`: See explanation of *custom property keys* ("[Custom Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`domain`: See explanation of *domains* ("[Subscriber Domains](#)" on page 2-6) in the *General API Concepts* (on page 2-1) chapter.

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

Return Value

None

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- `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.

For error codes description see Appendix A - List of Error Codes.

Examples

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

```
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

```
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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

- TRUE: if the subscriber was found in the database and successfully removed.
- FALSE: if the conditions for TRUE were not met: the subscriber was not found in the database, or the subscriber was found but was not successfully removed.

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST

For error codes description see Appendix A - List of Error Codes.

Example

- To remove subscriber *john* entirely from the database:

```
boolean isExist = removeSubscriber("john");
```

removeAllSubscribers

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

```
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

```
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

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN`
- `ERROR_CODE_DOMAIN_NOT_FOUND`

For error codes description see Appendix A - List of Error Codes.

getSubscriber

Syntax

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

Description

Retrieves 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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

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

Index 0	subscriber name (<code>java.lang.String</code>)
Index 1	array of mappings (<code>java.lang.String[]</code>)
Index 2	array of mapping types (<code>short[]</code>)
Index 3	domain name (<code>java.lang.String</code>)
Index 4	array of property names (<code>java.lang.String[]</code>)
Index 5	array of property values (<code>java.lang.String[]</code>)

- Index 6 array of custom property names (`java.lang.String[]`)
- Index 7 array of custom property values (`java.lang.String[]`)
- Index 8 auto-logout time, as seconds from now, or -1 if not set (`long[]`)

RPC Exception Error Codes

Following is the error code that may be returned by this method:

- `ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST`

For error codes description see Appendix A - List of Error Codes.

Example

To retrieve the subscriber record of *john*:

```
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

```
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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter

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

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

Description

Checks whether a subscriber that already exists in the SM database is logged in, that is, if the subscriber also exists in some SCE database.

Note that 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 some SCE database, but rather that 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 returned.

Parameters

`subscriberName`: See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

- TRUE: if the subscriber is logged in.
- FALSE: if the subscriber is not logged in.

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME

For error codes description see Appendix A - List of Error Codes.

getSubscriberNameByMapping

Syntax

```
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* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappingType`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`domain`: The name of the domain to which the subscriber belongs to. 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

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_DOMAIN_NOT_FOUND`
- `ERROR_CODE_BAD_SUBSCRIBER_MAPPING`
- `ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN`

For error codes description see Appendix A - List of Error Codes.

getSubscriberNames

Syntax

```
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.



Note

There is **no** guarantee that the total number of subscribers (in all bulks) will equal the value returned from `getNumOfSubscribers` at any time. They 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

Following is error code that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see Appendix A - List of Error Codes.

Example

```
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

```
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` ("[getSubscriberNames](#)" on page 3-20) operation.

Parameters

`lastBulkEnd`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) operation.

`numofSubscribers`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) 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 documentation of the Return Value section of the `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) operation.

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`
- `ERROR_CODE_DOMAIN_NOT_FOUND`

For error codes description see Appendix A - List of Error Codes.

getSubscriberNamesWithPrefix

Syntax

```
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` ("getSubscriberNames" on page 3-20) operation.

Parameters

`lastBulkEnd`: See description in `getSubscriberNames` ("getSubscriberNames" on page 3-20) operation.

`numOfSubscribers`: See description in `getSubscriberNames` ("getSubscriberNames" on page 3-20) 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 documentation of the Return Value section of the `getSubscriberNames` ("getSubscriberNames" on page 3-20) operation.

RPC Exception Error Codes

Following is error code that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see Appendix A - List of Error Codes.

getSubscriberNamesWithSuffix

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` ("getSubscriberNames" on page 3-20) operation.

Parameters

`lastBulkEnd`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) operation

`numOfSubscribers`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) 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 documentation of the Return Value section of the `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-20) operation.

RPC Exception Error Codes

Following is the error code that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see Appendix A - List of Error Codes.

getDomains

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

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

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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`properties`: See explanation of *property keys* ("[Subscriber Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

Return Value

None.

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`
- `ERROR_CODE_BAD_SUBSCRIBER_MAPPING`
- `ERROR_CODE_DOMAIN_NOT_FOUND`
- `ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST`
- `ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN`

For error codes description see Appendix A - List of Error Codes.

removeCustomProperties

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* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`CustomProperties`: See explanation of *custom property keys* ("[Custom Properties](#)" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

Return Value

None.

RPC Exception Error Codes

Following is the list of error codes that may be returned by this method:

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`
- `ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST`

For error codes description see Appendix A - List of Error Codes.

Blocking API Code Examples

This section gives 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.


```

package blocking;

import com.pcube.management.api.SMBlockingApi;

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

            //operations
            String[] domains=bapi.getDomains();
            int totalSubscribers=bapi.getNumberOfSubscribers();
            System.out.println(
                "number of subscribers in the database:\t\t "+
                totalSubscribers);
            for (int i=0; i<domains.length; i++) {
                int numberOfSubscribersInDomain=
                    bapi.getNumberOfSubscribersInDomain(domains[i]);
                System.out.println(
                    "number of subscribers domain "+domains[i]+
                    ":\t\t "+numberOfSubscribersInDomain);
            }
        } finally {
            //finalization
            bapi.disconnect();
        }
    }
}

```

Adding Subscriber, Printing Information, Removing 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.

```

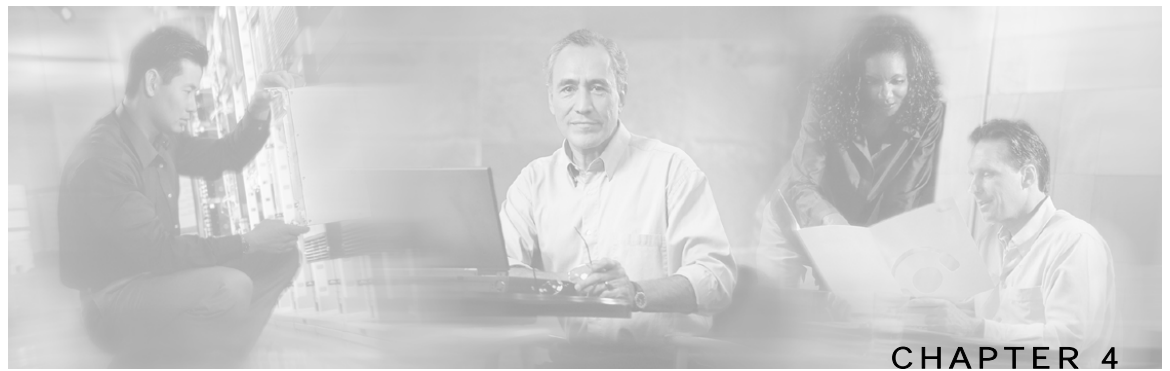
package blocking;

import com.pcube.management.api.SMBlockingApi;
import com.pcube.management.api.SMApiConstants;

public class AddPrintRemove {
    public static void main (String args[]) throws Exception {
        checkArguments(args);
        SMBlockingApi bapi = new SMBlockingApi();
        try {
            //initiation
            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[]{"custom-value"}, //custom property value
                args[5]); //domain
            //Print subscriber
            System.out.println("+ Printing subscriber");
            Object[] subfields = bapi.getSubscriber(args[1]);
            System.out.println("\tname:\t\t"+subfields[0]);
            System.out.println("\tmapping:\t"+
                ((String[])subfields[1])[0]);
            System.out.println("\tdomain:\t\t"+subfields[3]);
            System.out.println("\tautologout:\t"+subfields[8]);
            //Remove subscriber
            System.out.println("+ removing subscriber from SM");
            bapi.removeSubscriber(args[1]);
        } finally {
            //finalization
            bapi.disconnect();
        }
    }

    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);
        }
    }
}

```



Non-blocking API

This chapter introduces features unique to the Non-blocking API. It then lists all methods of the Non-blocking API, and ends with code examples.

This chapter contains the following sections:

- [Reliability Support](#) 4-1
- [Auto-reconnect Support](#) 4-2
- [Multi-threading Support](#) 4-2
- [ResultHandler Interface](#) 4-3
- [Non-blocking API Construction](#) 4-4
- [Non-blocking API Initialization](#) 4-5
- [Non-blocking API Methods](#) 4-6
- [Non-blocking API Code Examples](#) 4-8

Reliability Support

The Non-blocking API can work in two different modes, *reliable* and *non-reliable*, as described below. When the mode is not specified, the default is *reliable mode*.

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 were sent to the SM. Only after a reply from the SM is received, is the request 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 **non-committed** 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.

Non-reliable Mode

In non-reliable mode the API does not ensure that requests sent to the SM are executed. Also, all requests that are sent by the API when connection to the SM is down will be lost unless some external reliability mechanism is implemented.

Auto-reconnect Support

The Non-blocking API supports auto-reconnection 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 auto-reconnect support option can be activated regardless of the reliability mode.

Multi-threading Support

The Non-blocking API supports an unlimited number of threads calling its methods simultaneously.



Note

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

ResultHandler Interface

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

```
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`.

Example:

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

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



Note

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

```
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(...);
    }
}

```

Non-blocking API Construction

In addition to the constructors described in *API Construction* (on page 2-2), the Non-blocking API provides constructors that enable setting the reconnect period and the reliability mode.

Syntax:

The syntax for the additional Non-blocking API constructors is shown in the following code block:

```

public SMNonBlockingApi(long autoReconnectInterval)

public SMNonBlockingApi(boolean reliable, long autoReconnectInterval)

public SMNonBlockingApi(String legName, long autoReconnectInterval)

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

```

Arguments:

Following is a description of the constructor arguments for the additional Non-blocking API constructors:

- `autoReconnectInterval`
 Defines the interval (in milliseconds) for attempting reconnection by the reconnection task, as follows:

- value is 0 or less: the reconnection task is not activated (no auto-reconnect is attempted).
- value is greater than 0: in case of connection failure, the reconnection task will be activated every `<autoReconnectInterval>` milliseconds.

Default value: -1 (no auto-reconnect is attempted).



Note

In order to enable the auto-reconnect support, the `connect` method of the API **must** be activated at least once. For more information see, *Non-blocking API Code Examples* (on page 4-8).

- `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 non-reliable mode.

Default value: `TRUE` (the API works in reliable mode).

- `legName`

The name of the `LEG`, as described in *API Construction* (on page 2-2).

Examples:

The following code constructs a reliable API with an auto-reconnection interval of 10 seconds:

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

The following code constructs a reliable API without auto-reconnection support:

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI();

// Connect to the API
nbapi.connect(<SM IP address>);
```

The following code constructs a non-reliable API with auto-reconnection support:

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(false,10000);

// Initial connection - to enable the reconnect task
nbapi.connect(<SM IP address>);
```

Non-blocking API Initialization

The Non-blocking API enables initializing certain internal properties for API customization. This initialization is done using `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: a hint regarding the desired timeout (in milliseconds) on a non-responding PRPC protocol connection (the default is 45 seconds).

Syntax:

The syntax for the Non-blocking API `init` method is as follows:

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

Parameters:

Following is a description of the parameters for the Non-blocking API `init` method:

- `properties` (`java.util.Properties`)

Enables setting the properties described above:

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

Example:

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

```
// 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>);
```

Non-blocking API Methods

This section lists the methods of the Non-blocking API.

All methods return a handle of type *Long* that may be used to correlate operation calls and their results (see the *ResultHandler Interface* (on page 4-3) section).

The operation results passed to the result handler are the same as the return values described in the same method in the *Blocking API* ("[Blocking API Methods](#)" on page 3-3), except that:

- 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**

The result handler will be handed with an error **if and only if** the matching operation in the Blocking API would throw an exception with the same arguments according to the SM database state at the time of the call.

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

The following methods are described:

- `login`
- `logoutByName`
- `logoutByNameFromDomain`
- `logoutByMapping`
- `loginCable`
- `logoutCable`

login

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

logoutByName

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

logoutByNameFromDomain

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

logoutByMapping

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

loginCable

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

logoutCable

Syntax

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

The operation semantics are the same as the semantics of the matching Blocking API operation.

Non-blocking API Code Examples

This section gives 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("\tresult "+count+":\t"+tmp);
        }
    }

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

    //waits for result number 'last result' to arrive
    public synchronized void waitForLastResult(int lastResult) {
        while (count<lastResult) {
            try {
                wait(100);
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
        }
    }
}

public class LoginLogout {
    public static void main (String args[]) throws Exception{
        //check arguments
        checkArguments(args);
        int numSubscribersToLogin = Integer.parseInt(args[2]);

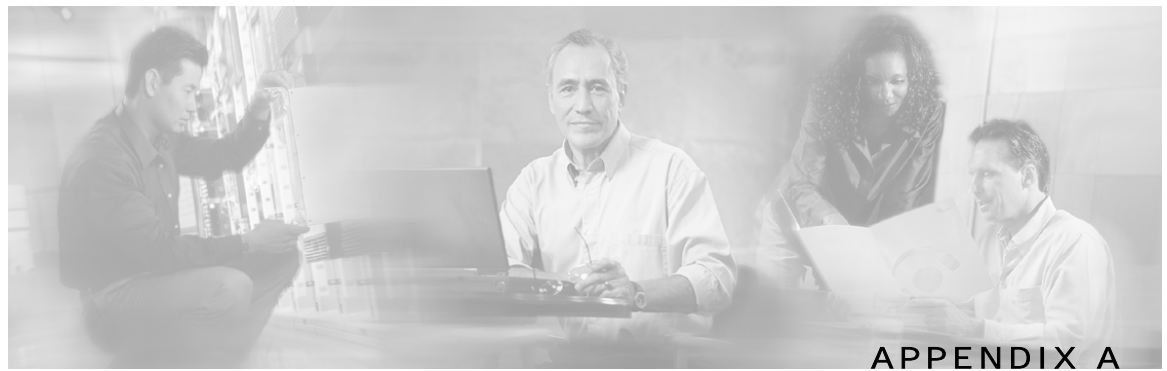
        //instantiation
        SMNonBlockingApi nbapi = new SMNonBlockingApi();
        try {
            //initiation
            nbapi.setDisconnectListener(
                new LoginLogoutDisconnectListener());
            nbapi.connect(args[0]);
            LoginLogoutResultHandler resultHandler =
                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,
                args[1], //domain
                false, //mappings are not additive
                -1); //disable auto-logout
        }
        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-susbcribers>");
        System.exit(1);
    }
}
//'automatic' mapping generator
private static String[] getMappings(int i) {
    return new String[]{ "10." +((int)i/65536)%256 + "." +
        ((int)(i/256))%256 + "." + (i%256)};
}
}
}

```

List of Error Codes

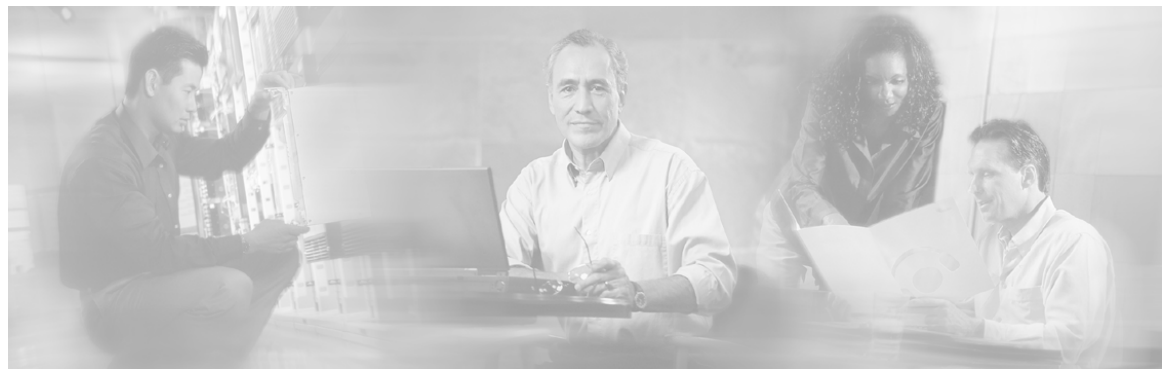
Error codes are used for interpreting the actual error for which an `RpcErrorException` was returned. The error code is extracted using the `getErrorCode` method.

The error code enumeration is given in the `com.pcube.management.api.SMApiConstants` interface. A list of the error codes and their description are given in the following table.

Table A-1 List of Error Codes

Error Code	Description
<code>ERROR_CODE_BAD_SUBSCRIBER_MAPPING</code>	A mapping was formatted badly or assigned to the subscriber illegally.
<code>ERROR_CODE_DOMAIN_NOT_FOUND</code>	The domain provided to the operation does not exist in the SM domain repository.
<code>ERROR_CODE_ILLEGAL_ARGUMENT</code>	One of the arguments provided to the method is illegal.
<code>ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME</code>	The subscriber name provided has more than 40 characters or has illegal characters.
<code>ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN</code>	The domain provided to the operation exists in the SM domain repository but is not a subscriber domain.
<code>ERROR_CODE_NUMBER_FORMAT</code>	A VLAN mapping string provided to the API does not represent a decimal number.
<code>ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST</code>	The subscriber on which the operation is performed does not exist in the SM database.
<code>ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION</code>	The subscriber exists in the SM database but is associated with a domain other than the one specified by the operation.
<code>ERROR_CODE_SUBSCRIBER_MAPPING_CONGESTION</code>	The mappings provided for the subscriber by the operation already belong to another subscriber.
<code>ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS</code>	The subscriber on which the operation was performed already exists in the SM database.
<code>ERROR_CODE_ARRAY_ACCESS</code>	Internal SM error.
<code>ERROR_CODE_ATTRIBUTE_NOT_FOUND</code>	Internal SM error.

Error Code	Description
ERROR_CODE_CLASS_CAST	Internal SM error.
ERROR_CODE_CLASS_NOT_FOUND	Internal SM error.
ERROR_CODE_CLIENT_INTERNAL_ERROR	Internal error.
ERROR_CODE_CLIENT_OUT_OF_THREADS	Internal error.
ERROR_CODE_ILLEGAL_STATE	Internal SM error.
ERROR_CODE_OBJECT_NOT_FOUND	Internal SM error.
ERROR_CODE_OPERATION_NOT_FOUND	Internal SM error.
ERROR_CODE_OUT_OF_MEMORY	Internal SM error.
ERROR_CODE_RUNTIME	Internal SM error.
ERROR_CODE_NULL_POINTER	Internal SM error.
ERROR_CODE_SE_ERROR	Internal SM error. The SM could not perform the operation on the SCE device.
ERROR_CODE_UNKNOWN	Internal SM or API error.



Index

A

Adding Subscriber, Printing Information,
 Removing Subscriber • 3-27
addSubscriber • 3-13
API Construction • 2-2
API Finalization • 2-4
API Initialization • 2-2
Audience • v
Auto-reconnect Support • 4-2

B

Blocking API • 2-1, 3-1
 loginCable • 3-10
 logoutByMapping • 3-9
 logoutByName • 3-7
 logoutByNameFromDomain • 3-8
 logoutCable • 3-12
Blocking API Code Examples • 3-26
Blocking API Methods • 3-3
Blocking API setup • 2-3
Blocking API/Non-blocking API • 2-1

C

Cisco TAC Website • vi
Compiling and running • 1-3
Connecting to the SM • 2-3
Constructor that accepts a LEG name • 2-2
Custom Properties • 2-6

D

Description • 3-4, 3-7, 3-8, 3-9, 3-11, 3-12,
 3-13, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20,
 3-22, 3-23, 3-24, 3-25
DisconnectListener Interface • 2-7
Document Conventions • vi

E

Example • 3-8, 3-9, 3-10, 3-15, 3-18, 3-21
Example: • 3-13
Examples • 3-6, 3-12, 3-13, 3-15
Exceptions • 2-7
Extracting the Package • 1-2

G

General API Concepts • 2-1
getDomains • 3-24
getNumberOfSubscribers • 3-16
getNumberOfSubscribersInDomain • 3-16
getSubscriber • 3-17
getSubscriberNameByMapping • 3-19
getSubscriberNames • 3-20
getSubscriberNamesInDomain • 3-22
getSubscriberNamesWithPrefix • 3-22
getSubscriberNamesWithSuffix • 3-23
Getting Number of Subscribers • 3-26
Getting Started • 1-1

I

Installation • 1-2
Introduction • 1-1

L

List of Error Codes • A-1
login • 3-4, 4-7
Login and Logout • 4-9
loginCable • 3-10, 4-8
loginCable method
 blocking API • 3-10
logoutByMapping • 3-9, 4-8
logoutByMapping method
 blocking API • 3-9
logoutByName • 3-7, 4-7
logoutByName method

- blocking API • 3-7
- logoutByNameFromDomain • 3-8, 4-8
- logoutByNameFromDomain method
 - blocking API • 3-8
- logoutCable • 3-12, 4-8
- logoutCable method
 - blocking API • 3-12

M

- Multi-threading Support • 3-1, 4-2

N

- Network ID Mappings • 2-4
- Non-blocking API • 2-2, 4-1
- Non-blocking API Code Examples • 4-8
- Non-blocking API Construction • 4-4
- Non-blocking API Initialization • 4-5
- Non-blocking API Methods • 4-6
- Non-blocking API setup • 2-3
- Non-reliable Mode • 4-2

O

- Opening a TAC Case • vii

P

- Package Content • 1-2
- Parameters • 3-5, 3-7, 3-8, 3-9, 3-11, 3-12, 3-14, 3-15, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-23, 3-24, 3-25, 3-26
- Platforms • 1-1
- Preface • v
- Purpose • v

R

- Related Publications • v
- Reliability Support • 4-1
- Reliable Mode • 4-2
- removeAllSubscribers • 3-16
- removeCustomProperties • 3-25
- removeSubscriber • 3-15
- ReplyTimeout and OperationTimeout
 - Exception • 3-2
- ResultHandler Interface • 4-3
- Return Value • 3-6, 3-7, 3-8, 3-10, 3-11, 3-12, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-23, 3-24, 3-25, 3-26
- RPC Exception Error Codes • 3-5, 3-7, 3-9, 3-10, 3-11, 3-12, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-23, 3-24, 3-25, 3-26

S

- setPropertyToDefault • 3-25
- Setup Operations • 2-3
- SM setup • 1-3
- Specifying IP Address Mapping • 2-5
- Specifying IP Range Mapping • 2-5
- Specifying VLAN Tag Mapping • 2-5
- Subscriber Domains • 2-6
- Subscriber Name Format • 2-4
- Subscriber Properties • 2-6
- subscriberExists • 3-18
- subscriberLoggedIn • 3-19
- Syntax • 3-4, 3-7, 3-8, 3-9, 3-10, 3-12, 3-13, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-22, 3-23, 3-24, 3-25, 4-7, 4-8

T

- TAC Case Priority Definitions • vii
- Technical Support • vi