



## **SCMS SM C / C++ API Programmer's Guide**

OL-7203-01

**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: DOCOL-7203-01=  
Text Part Number: OL-7203-01



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 C / C++ API Programmer's Guide*

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



## **Preface v**

Audience v

Purpose v

Related Publications v

Document Conventions v

Technical Support vi

    Cisco TAC Website vi

    Opening a TAC Case vi

    TAC Case Priority Definitions vii

## **Getting Started 1-1**

Platforms and Compilers 1-1

Installation 1-1

    Extracting the Package 1-1

Compiling and Running 1-3

    Windows 1-3

    Solaris 1-3

    Linux (Red Hat) 1-4

    SM Setup 1-4

## **General API Concepts 2-1**

Blocking API/Non-blocking API 2-1

    Blocking API 2-2

    Non-blocking API 2-2

Reliability 2-2

C vs. C++ API 2-2

    Method Names 2-2

    Handle Pointers 2-3

API Initialization 2-3

- API Construction 2-4
- Setup Operations 2-5
- Connecting to the SM 2-5
- API Finalization 2-5
- Return Code Structure 2-5
  - Definitions 2-6
  - Example 2-7
- Error Code Structure 2-7
  - Definition 2-7
- Subscriber Name Format 2-7
- Network ID Mappings 2-8
  - Specifying IP Address Mapping 2-8
  - Specifying IP Range Mapping 2-9
  - Specifying VLAN Tag Mapping 2-9
- Subscriber Domains 2-9
- Subscriber Properties 2-10
- Custom Properties 2-10
- Logging Capabilities 2-10
- Disconnect Callback Listener 2-11
- Signal Handling 2-11
- Blocking API 3-1**
  - Multi-threading Support 3-1
  - Operation Timeout Error Code 3-2
  - Blocking API Methods 3-3
    - login 3-5
    - logoutByName 3-8
    - logoutByNameFromDomain 3-9
    - logoutByMapping 3-11
    - loginCable 3-12
    - logoutCable 3-14
    - addSubscriber 3-15
    - removeSubscriber 3-17
    - removeAllSubscribers 3-18

- getNumberOfSubscribers 3-18
- getNumberOfSubscribersInDomain 3-18
- getSubscriber 3-19
- subscriberExists 3-20
- subscriberLoggedIn 3-21
- getSubscriberNameByMapping 3-22
- getSubscriberNames 3-22
- getSubscriberNamesInDomain 3-24
- getSubscriberNamesWithPrefix 3-25
- getSubscriberNamesWithSuffix 3-26
- getDomains 3-27
- setPropertyToDefault 3-27
- removeCustomProperties 3-28
- C++ setLogger Method 3-28
- C++ init Method 3-29
- C SMB\_init Function 3-30
- C SMB\_release Function 3-31
- setName 3-31
- connect 3-31
- disconnect 3-32
- isConnected 3-32
- Blocking API C++ Code Examples 3-32
  - Getting Number of Subscribers 3-33
  - Adding Subscriber, Printing Information, Removing Subscriber 3-33
- Blocking API C Code Examples 3-35
  - Getting Number of Subscribers 3-35
  - Adding Subscriber, Printing Information, Removing Subscriber 3-37

### **Non-blocking API 4-1**

- Multi-threading Support 4-1
- Result Handler Callbacks 4-2
- Non-blocking API Methods 4-4
  - login 4-5
  - logoutByName 4-5

- logoutByNameFromDomain 4-5
- logoutByMapping 4-5
- loginCable 4-6
- logoutCable 4-6
- C++ setLogger Method 4-6
- C++ init Method 4-6
- C SMNB\_init Function 4-6
- C SMNB\_release Function 4-7
- setName 4-7
- connect 4-7
- disconnect 4-7
- isConnected 4-7
- Non-blocking API C++ Code Examples 4-7
  - Login and Logout 4-8
- Non-blocking API C Code Examples 4-10
  - Login and Logout 4-10

**List of Error Codes A-1**

**Index 1**



## Preface

---

The C/C++ API is used for updating, querying, and configuring the SM Manager. It consists of two parts, which may be used separately or together without restriction.

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

## 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 C / C++ 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.
<b>Bold</b>	Names of menus, options, and command buttons.
Courier	System output shown on the computer screen in the Telnet session.

Typeface or Symbol	Meaning
<b>Courier Bold</b>	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> (<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 considers the following aspects of the SM C/C++ API: on which platforms can it be used, and how to install, compile, and start running it.

This chapter contains the following sections:

- [Platforms and Compilers](#) 1-1
- [Installation](#) 1-1
- [Compiling and Running](#) 1-3

## Platforms and Compilers

The SM C/C++ API was developed and tested on Windows, Solaris, and Linux platforms. It was compiled on Windows using Microsoft Visual C++ 6 compiler, on Solaris using the GCC 2.95.3 compiler, and on Linux using GCC 3.2.3 compiler.

## Installation

### Extracting the Package

The C/C++ 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-c-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 C/C++ SM API version and build number.

## Package Content

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

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

The `<installdir>/winnt` folder contains the `smapi.dll` file, which is the Windows API Executable. It also contains additional DLL and LIB files necessary for the Windows API operation.

The `<installdir>/solaris` folder contains the `libsmapi.so` file, which is the Solaris API Executable.

The `<installdir>/linux` folder contains the `libsmapi.so` file, which is the Linux API Executable.

The `<installdir>/include` folder contains the API C/C++ header files.

The `<installdir>/include/system` folder contains some C++ API internal header files.

**Table 1-1** Layout of Installation Directory

Folder	Subfolder (as applicable)	File
<code>&lt;installdir&gt;</code>		README
	<code>doc</code>	<code>C_SM_API_User_Manual.pdf</code>
	<code>include</code>	BasicTypes.h Common.h GeneralDefs.h Logger.h PrintLogger.h SmApiBlocking.h SmApiBlocking_c.h SmApiNonBlocking.h SmApiNonBlocking_c.h
	<code>include/System</code>	OperationHandleInterface.h OperationResultInterface.h
	<code>linux</code>	<code>libsmapi.so</code>
	<code>solaris</code>	<code>libsmapi.so</code>

Folder	Subfolder (as applicable)	File
	Winnt	
		asn1ber.dll
		asn1ber.lib
		asn1rt.dll
		asn1rt.lib
		SmApi.dll
		SmApi.lib

## Compiling and Running

### Windows

To compile and run a program that uses the SM C/C++ API, do the following:

---

**Step 1** Have `smapi.dll` and the other DLL files in your `PATH` or in the directory of your executable.

**Step 2** Have the `include` folder in your include path of the compilation.

**Example using Microsoft Visual C++ 6:**

Enter the project settings, choose the **C++** tab, and then choose the **Preprocessor** category. Add the include directory path in the **Additional Include directories** line.

**Step 3** Have the `smapi.lib` file in you linkage path.

**Example using Microsoft Visual C++ 6:**

Enter the project settings, and choose the **Link** tab. Add `smapi.lib` to the **Object\Library modules** line.

**Step 4** Include the relevant API header file in your source code, and compile your code.

---

### Solaris

To compile and run a program that uses the SM C/C++ API, do the following:

---

**Step 1** Have `libsmapi.so` in your `LD_LIBRARY_PATH`.

For example, when using the **Bash** shell type, use the following command line:

```
bash-2.03$ export set LD_LIBRARY_PATH=$LD
LIBRARY_PATH:the-libsmapi.so-folder
```

**Step 2** Have the `include` folder in your include path of the compilation.

For example, when using the **GCC**, add the include folder after the `-I` option flag, as follows:

```
> gcc -c -o TestSmApi.o -Ism-api-header-file-folder
-Ism-api-header-file-folder/system/ TestSmApi.cpp
```

**Step 3** Have the `libsmapi.so` file in your linkage line or load it dynamically.

Link your object file to the *pthread* and *socket* library, as follows:

```
> gcc -o testSmApi TestSmApi.o -lsmapi -lpthread -lsocket
```

## Linux (Red Hat)

To compile and run a program that uses the SM C/C++ API, do the following:

**Step 1** Have `libsmapi.so` in your `LD_LIBRARY_PATH`.

For example, when using the **Bash** shell type, use the following command line:

```
bash-2.03$ export set LD_LIBRARY_PATH=$LD
LIBRARY_PATH:the-libsmapi.so-folder
```

**Step 2** Have the `include` folder in your include path of the compilation.

For example, when using the **GCC**, add the include folder after the `-I` option flag, as follows:

```
> gcc -c -o TestSmApi.o -Ism-api-header-file-folder
-Ism-api-header-file-folder/system/ TestSmApi.cpp
```

**Step 3** Have the `libsmapi.so` file in your linkage line or load it dynamically. Specify the location of `libsmapi.so` using the `-L` option flag. Link your object *file* to the *pthread* and *stdc++* libraries, as follows:

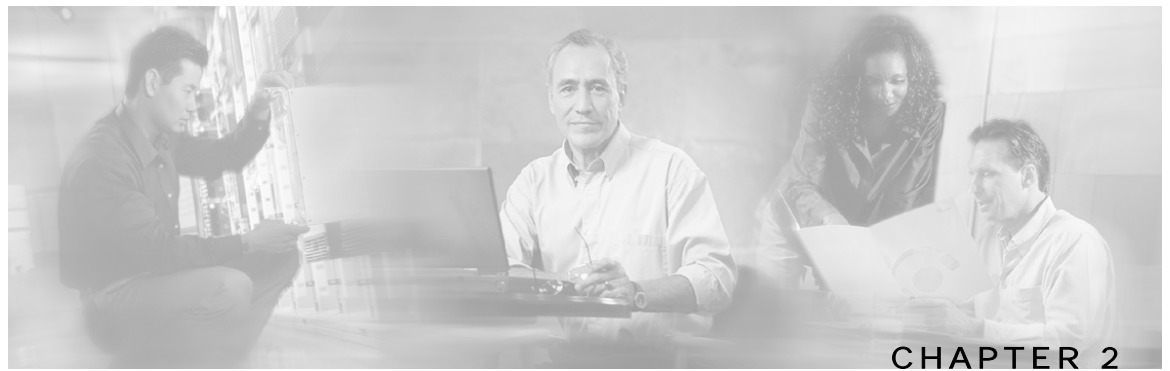
```
> gcc -o testSmApi TestSmApi.o -lsmapi -lpthread -lstdc++ -
L<lib path>
```

## SM Setup

The API connects to the PRPC server on the SM. For the API to work, the following conditions must be met:

- 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 additional information, see the *SCMS Subscriber Manager User Guide*.



## General API Concepts

---

This chapter describes various concepts that are utilized when working with the SM C/C++ API.

This chapter contains the following sections:

- [Blocking API/Non-blocking API](#) 2-1
- [Reliability](#) 2-2
- [C vs. C++ API](#) 2-2
- [API Initialization](#) 2-3
- [API Finalization](#) 2-5
- [Return Code Structure](#) 2-5
- [Error Code Structure](#) 2-7
- [Subscriber Name Format](#) 2-7
- [Network ID Mappings](#) 2-8
- [Subscriber Domains](#) 2-9
- [Subscriber Properties](#) 2-10
- [Custom Properties](#) 2-10
- [Logging Capabilities](#) 2-10
- [Disconnect Callback Listener](#) 2-11
- [Signal Handling](#) 2-11

### Blocking API/Non-blocking API

This section 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 C/C++ API provides a wide range of operations and is a *superset* of the Non-blocking API functionality.

## Non-blocking API

In a Non-blocking API, every method returns *immediately*, even before its operation has been completed. The operation results are either returned to a set of user defined callbacks 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 C/C++ API contains a small number of Non-blocking operations. The API supports retrieval of operation results using result callbacks.

## Reliability

The API is reliable in the sense that the operations performed on the API are kept till their associated results arrive from the SM. If the connection to the SM is lost, operations that were not sent to the SM and operations whose results did not yet arrive from the SM are sent to the SM immediately on reconnection. The order of operation invocation is maintained at all times.

## C vs. C++ API

The C and C++ APIs are essentially the same. The C API is actually a thin wrapper over the C++ API, with method prototype and signature changes imposed by the constraint of C not being an object-oriented programming language.

Following is a list of C/C++ API differences and some examples.

## Method Names

The method names of the C API are the same as in the C++ API, except that the C API method names have an identifying prefix:

- The Blocking C API method names are prefixed with `SMB_`.
- The Non-blocking C API methods are prefixed with `SMNB_`.



### Note

The documentation of methods in this guide, uses the name and signature of the C++ API methods.



## Example

Both the Blocking and Non-blocking C++ APIs provide a `login` method. The same method in the Blocking C API is called `SMB_login` and in the Non-blocking C API is called `SMNB_login`.

## Handle Pointers

The Blocking and Non-blocking APIs are C++ Objects. Several API instances can co-exist in a single process, each having its own socket and state. To provide the same functionality in the C API, the *handle* concept is introduced. Each C API method accepts a handle as its first argument. The handle is used to identify the C API instance on which the caller wants to operate. Calling the `SMB_init` or `SMNB_init` method creates a new C API instance; the handle to the instance is provided by the return value of these methods. For more information, see *API Initialization* (on page 2-3).

## Blocking API Example

The following C++ Blocking API `logoutByName` method signature:

```
ReturnCode* logoutByName(      char* argName,
                              char** argMappings,
                              MappingType* argMappingTypes,
                              int argMappingsSize)
```

is translated to

```
ReturnCode* SMB_logoutByName(  SMB_HANDLE argApiHandle,
                              char* argName,
                              char** argMappings,
                              MappingType* argMappingTypes,
                              int argMappingsSize);
```

## Non-Blocking API example

The following C++ Non-blocking API `logoutByName` method signature:

```
int logoutByName(char* argName,
                 char** argMappings,
                 MappingType* argMappingTypes,
                 int argMappingsSize);
```

is translated to

```
int SMNB_logoutByName(        SMNB_HANDLE argApiHandle,
                              char* argName,
                              char** argMappings,
                              MappingType* argMappingTypes,
                              int argMappingsSize);
```

## API Initialization

To initialize the API:

- 
- Step 1** Construct the API using one of its two constructors.
  - Step 2** Perform API-specific setup operations.

**Step 3** Connect the API to the SM.

These three steps are described in the following sections.

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

## API Construction

Both C and C++ Blocking and Non-blocking APIs must construct and initialize the API. Be sure to check that the initialization succeeded before proceeding.

**Step 1** For the C++ API, construct an API object, and then call its `init` method. For example:

```
SmApiNonBlocking nbapi;
If (!nbapi.init(0,2000000,10,30)){
    exit(1);
}
```

**Step 2** For the C API, call the `init` function, which allocates and initializes the API. For example:

```
SMNB_HANDLE nbapi = SMNB_init(0,2000000,10,30);
if (nbapi == NULL) {
    exit(1);
}
```

## SETTING the 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*.

To set the LEG name, call the relevant `setName` function in the API. 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.C`
- For Non-blocking API: `.NB.SM-API.C`

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.C`

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

For additional information about LEG-SM failure handling, see the *Configuration File Options Appendix* of the *SCMS Subscriber Manager User Guide*.

## Setup Operations

The setup operations differ for the two APIs. Both APIs support setting a disconnect listener, which is described in the *Disconnect Callback Listener* (on page 2-11) section.

### Blocking API setup

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

### Non-blocking API setup

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

## Connecting to the SM

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

- The following example shows how to connect when using the C++ APIs:

```
connect(char* host, Uint16 argPort = 14374)
```

- The following example shows how to connect when using the C Blocking API:

```
SMB_connect(SMB_HANDLE argApiHandle, char* host, Uint16 argPort)
```

- The following example shows how to connect when using the C Non-blocking API:

```
SMNB_connect(SMNB_HANDLE argApiHandle, char* host, Uint16 argPort)
```

The `argHostName` 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 one of the variants of the function `isConnected`.

## API Finalization

Both C and C++ Blocking and Non-blocking APIs must disconnect from the SM and free the memory of the API:

- For the C++ APIs, call the `disconnect` method and free the API object.
- For the C APIs, call the appropriate `disconnect` function and then free the API using the appropriate `release` function.

## Return Code Structure

The Results of the API operations are returned using a generic structure called `ReturnCode`. The `ReturnCode` structure contains several parameters:

- `u` - a union of all of variables and pointers to variables that are the actual returned value.
- `type` - a return code type parameter (`ReturnCodeType` enumeration) that defines the type of the value (`u`) that the `ReturnCode` structure holds.

- `size` - the number of elements in the value. If `size` equals 1 then there is one element in the value, such as a scalar, character string, void, or error. If `size` is greater than 1 then there are several elements in the array, and the type should be one of the array types.

The return code structure is allocated by the API, and must be freed by the user of the API. The `freeReturnCode` utility function can be used to safely free the structure.

Additional return code structure utility functions are: `printReturnCode` that prints the `ReturnCode` structure value to the `stdout`, and `isReturnCodeError` that checks whether the `ReturnCode` structure is an error.

## Definitions

From `GeneralDefs.h` header file:

```
OSAL_DllExport typedef struct ReturnCode_t
{
    ReturnCodeType type;
    int size;          /* number of elements in the union element (for
example:
stringVal will have size=1) */
    union {            /* use union value according to the type
value */
        bool boolVal;
        short shortVal;
        int intVal;
        long longVal;
        char* stringVal;
        bool* boolArrayVal;
        short* shortArrayVal;
        int* intArrayVal;
        long* longArrayVal;
        char** stringArrayVal;
        ErrorCode* errorCode;
        struct ReturnCode_t** objectArray;
    } u;
}ReturnCode;
```

## Example

In the following example the subscriber data of **subscriber1** is retrieved and displayed. The returned structure contains an array of `ReturnCode` structures held by the `objectArray` union value. Each structure contains a different value type. For additional information, see the explanation of the `getSubscriber` method ("[getSubscriber](#)" on page 3-19) in *the Blocking API* (on page 3-1) chapter. Notice that the example code uses the methods `isReturnCodeError` and `freeReturnCode`.

```
ReturnCode* subFields = bapi.getSubscriber("subscriber1");
if (isReturnCodeError(subFields) == false)
{
    printf("\tname:\t\t%s\n", subFields->u.objectArray[0]->u.stringVal);
    printf("\tmapping:\t\t%s\n",
           subFields->u.objectArray[1]->u.stringArrayVal[0]);
    printf("\tdomain:\t\t%s\n", subFields->u.objectArray[3]->u.stringVal);
    printf("\tautologout:\t\t%d\n", subFields->u.objectArray[8]->u.intVal);
}
else
{
    printf("error in subscriber retrieval\n");
}
freeReturnCode(subFields);
```

## Error Code Structure

The `ErrorCode` structure can be one of the values of the return code structure. This structure describes the error that occurred. The structure consists of the following parameters:

- `type` - an `ErrorCodeType` enumeration that describes the error; see the `GeneralDefs.h` file for additional information.
- `message` - a specific error code.
- `name` - currently not used.

## Definition

From `GeneralDefs.h` header file:

```
OSAL_DllExport typedef struct ErrorCode_t
{
    ErrorCodeType type; /* type of the error see enumeration */
    char* name;         /* currently not used */
    char* message;     /* error message */
}ErrorCode;
```

## 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 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 APIs)
- the `login` method (Blocking or Non-blocking APIs)

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

- A character string (`char*`) mapping identifiers or array of character strings (`char**`) mappings.
- A `MappingType` enumeration or array of `MappingType` variables.

When passing arrays, the `MappingType` variables array must contain the same amount of elements as the mappings array.

The API supports the following subscriber mapping types (defined by the `MappingType` enumeration):

- 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 `GeneralDefs.h` header file:

- `IP_RANGE` specifies IP mapping (IP-Address or IP-Range that matches the mapping identifier with the same index in the mapping identifier array).

## 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 `GeneralDefs.h` header file:

- `VLAN` specifies a VLAN mapping that matches the mapping identifier with the same index in the mapping identifier array.

## 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 `(char*)`. 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 `ERROR_CODE_DOMAIN_NOT_FOUND` error `ReturnCode` 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 C/C++ API operations, use the String arrays (`char**`) `propertyKeys` and `propertyValues`. Note that 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]`.

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 (`char**`) arrays `customPropertyKeys` and `customPropertyValues`, the same as used in formatting *Subscriber Properties* (on page [2-10](#)).

## Logging Capabilities

The API package contains a `Logger` abstract class that can be inherited and used to integrate the SM API with the host application's log. The `Logger` class exposes four basic levels of logging: error messages, warning messages, informative messages, and several levels of trace messages. Both the Blocking and Non-blocking API have this capability. The `Logger` class is provided in the `Logger.h` header file.

The API user should implement a logger by inheriting from the `Logger` class. To have the API use this logger, the code should call the API's `setLogger` method of the C++ implementation of the API.

For testing and for simple LEG implementations, the API package provides the `PrintLogger` class, which is a simple implementation of the `Logger` class that prints the log messages to the standard error (STDERR). An API user can initiate the `PrintLogger` object, set its logging level using the `setLoggingLevels` method of the `PrintLogger` class, and pass the logger object to the API using the API's `setLogger` method. The `PrintLogger` class is provided in the `PrintLogger.h` header file.



## Disconnect Callback Listener

Both APIs (Blocking and Non-blocking) allow setting a disconnect callback listener. The disconnect callback is defined as follows:

```
typedef void (*ConnectionIsDownCallBackFunc)();
```

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

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

Example:

- Following is a simple implementation of a disconnect callback listener that prints a message to `stdout` and exits.

```
#include "GeneralDefs.h\n\nvoid connectionIsDown(){\n    printf("Message: connection is down.");\n    exit(0);\n}
```

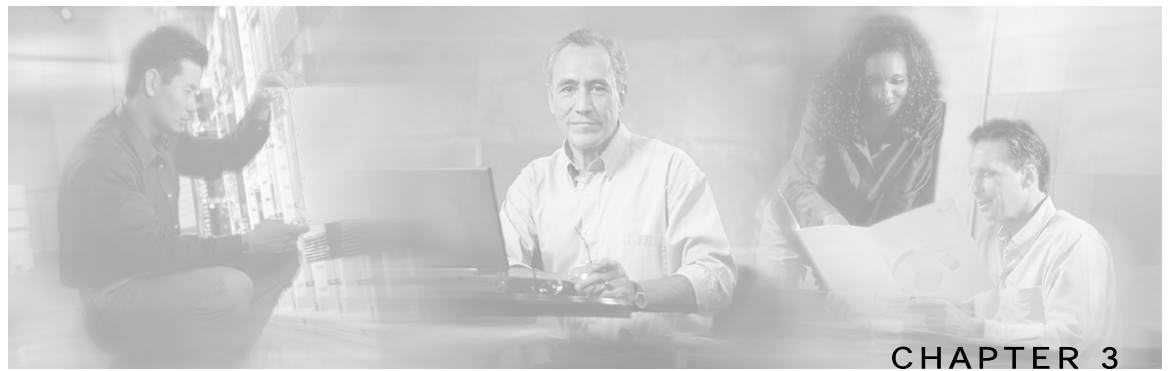
## Signal Handling

The SM API as a networking module might handle sockets that are closed by the SM - for example, if the SM is restarted - which may cause “Broken Pipe” signals. It is especially advisable for the Unix environment to handle this signal.

For example, to ignore the signal add the following call:

```
sigignore(SIGPIPE);
```





## Blocking API

---

This chapter introduces the Reply Timeout, a feature unique to the Blocking API. It then lists all operations of the Blocking API, and ends with some code examples.



---

**Note**

If you want 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
- [Operation Timeout Error Code](#) 3-2
- [Blocking API Methods](#) 3-3
- [Blocking API C++ Code Examples](#) 3-32
- [Blocking API C Code Examples](#) 3-35

## Multi-threading Support

The Blocking API supports a configurable number of threads calling its methods simultaneously. For more information about configuring the number of threads, see the `init` ("C++ [init Method](#)" on page 3-29) method.



---

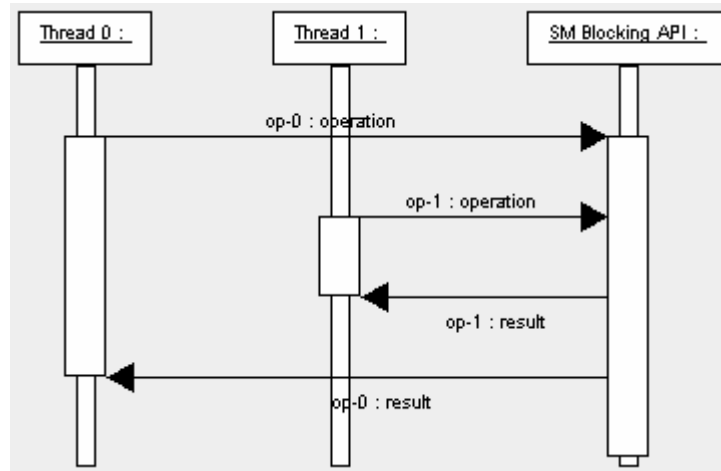
**Note**

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):



## Operation Timeout Error Code

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 return a `ReturnCode` with the `ERROR_CODE_CLIENT_OPERATION_TIMEOUT` error when a reply does not return within the timeout period.

Calling the `setReplyTimeout` function with an `int` 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.

## 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.

The C and C++ API share the same function signature, except for the `SMB_` prefix for all function names of the Blocking C APIs, and the API handle of type `SMB_HANDLE` as the first parameter in all functions. Any other differences between the APIs are explained in the function description.

The Blocking API subscriber management methods 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:

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



---

**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.

---

### **Note, Static/Manual Subscriber configuration.**

For example for GUI usage, the following methods are relevant:

- addSubscriber
- removeSubscriber
- removeAllSubscribers
- setPropertiesToDefault
- removeCustomProperties

### **For simple read-only operations performed independently in subscriber awareness mode.**

The following methods are relevant:

- getNumberOfSubscribers
- getNumberOfSubscribersInDomain
- getSubscriber
- subscriberExists

- subscriberLoggedIn
- getSubscriberNameByMapping
- getSubscriberNames
- getSubscriberNamesInDomain
- getSubscriberNamesWithPrefix
- getSubscriberNamesWithSuffix
- getDomains

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

**Methods used for API maintenance - initialization, connection etc.:**

- *C++ setLogger method* (on page 3-28)
- *C++ init method* (on page 3-29)
- *C SMB\_init function* (on page 3-30)
- *C SMB\_release function* (on page 3-31)
- *setName* (on page 3-31)
- *connect* (on page 3-31)
- *disconnect* (on page 3-32)
- *isConnected* (on page 3-32)




---

**Note**

The examples that appear at the end of the described methods are in C++.

---




---

**Note**

Every example described at the end of the methods should be preceded by the sample code below:

---

```
SmApiBlocking bapi;
// Init with default parameters
bapi.init();
// Connect to the SM
bapi.connect((char*)"1.1.1.1");
```

## login

**Syntax**

```

ReturnCode* login(      char* argName,
                       char** argMappings,
                       MappingType* argMappingTypes,
                       int argMappingsSize,
                       char** argPropertyKeys,
                       char** argPropertyValues,
                       int argPropertySize,
                       char* argDomain,
                       bool argIsAdditive,
                       int argAutoLogoutTime)

```

**Description**

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

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

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

**Parameters**

`argName`: See explanation of *subscriber name format* (on page 2-7) in the *General API Concepts* chapter.

`argMappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

If no mappings are specified, and the `argIsAdditive` flag is TRUE, the previous mappings will be retained. If no such mappings exist, the operation will fail.

`argMappingTypes`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

`argMappingsSize`: the size of the `argMappings` and `argMappingTypes` arrays.

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

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

`argPropertySize`: the size of the `argPropertyKeys` and `argPropertyValues` arrays.

`argDomain`: See explanation of *domains* ("[Subscriber Domains](#)" on page 2-9) in the *General API Concepts* chapter.

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

`ArgIsAdditive`: Refers to the mappings parameters.

- TRUE: adds the mappings provided by this call to the subscriber record.

- `FALSE`: overrides the mappings that already exist in the subscriber record with the mappings provided by this call.

`argAutoLogoutTime`: 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.

## Return Value

A pointer to a `ReturnCode` structure with a void type, unless an error occurred.

## 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` this error 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* ("[List of Error Codes](#)" on page [A-1](#)).



## Example

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

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

bapi.login(
    "john",                // subscriber name
    &ip_address,
    &map_type,
    1,                    // one mapping
    NULL, NULL, 0,        // no properties
    "subscribers",        // domain
    true,                 // isMappingAdditive is true
    -1);                 // autoLogoutTime set to infinite
```

To add the IP address 192.168.12.5 overriding previous mappings:

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

bapi.login(
    "john",                // subscriber name
    &ip_address,
    &map_type,
    1,
    NULL, NULL, 0,
    "subscribers",        // domain
    false,                 // isMappingAdditive is false
    -1);                 // autoLogoutTime set to infinite
```

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

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

bapi.login(
    "john",                // subscriber name
    &ip_address,
    &map_type,
    1,
    NULL, NULL, 0,
    "subscribers",        // domain
    false,                 // isMappingAdditive is false
    300);                 // autoLogoutTime set to 300 seconds
```

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

```
char* prop_name = "packageID";
char* prop_value = "10";

bapi.login(
    "john",
    NULL, NULL, 0,
    &prop_name,            // property key
    &prop_value,          // property value
    1,                    // one property
    "subscribers",        // domain
    false,
    -1);
```

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

```

MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

char* prop_name = "packageID";
char* prop_value = "10";

bapi.login(
    "john",
    &ip_address,
    &map_type,
    1,
    &prop_name,           // property key
    &prop_value,         // property value
    1,
    "subscribers",      // domain
    true,                // isMappingAdditive is set to true
    -1);

```

## logoutByName

### Syntax

```

ReturnCode* logoutByName(   char* argName,
                           char** argMappings,
                           MappingType* argMappingTypes,
                           int argMappingsSize)

```

### Description

Locates the subscriber in the database and removes mappings from it.

### Parameters

**argName:** See explanation of *subscriber name format* (on page 2-7) in the *General API Concepts* chapter.

**argMappings:** See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

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

**argMappingTypes:** See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

**argMappingsSize:** The size of the `argMappings` and `argMappingTypes` arrays.

### Return Value

A pointer to a `ReturnCode` structure with a Boolean type:

- 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.

## 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 *List of Error Codes* (on page [A-1](#)).

## Example

To remove IP address 192.168.12.5 of subscriber *john*:

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

bapi.logoutByName(
    "john",
    &ip_address,
    &map_type,
    1);
```

To remove all IP addresses of subscriber *john*:

```
bapi.logoutByName("john", NULL, NULL, 0);
```

## logoutByNameFromDomain

### Syntax

```
ReturnCode* logoutByNameFromDomain (char* argName,
                                     char** argMappings,
                                     MappingType* argMappingTypes,
                                     int argMappingsSize,
                                     char* argDomain)
```

### Description

Locates the subscriber in the database according to the specified domain and removes mappings from it.

## Parameters

`argName`: See explanation of *subscriber name format* (on page 2-7) in the *General API Concepts* chapter.

`argMappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

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

`argMappingTypes`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

`argMappingsSize`: The size of the `argMappings` and `argMappingTypes` arrays.

`argDomain`: See explanation of *domains* ("[Subscriber Domains](#)" on page 2-9) in the *General API Concepts* 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

A pointer to a `ReturnCode` structure with a Boolean type:

- 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.

## 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 List of Error Codes.

## Example

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

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";

bapi.logoutByNameFromDomain(
    "john",
    &ip_address,
    &map_type,
    1,
    "subscribers");
```

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

```
bapi.logoutByNameFromDomain(
    "john",
    NULL,
    NULL,
    0,
    "subscribers");
```

## logoutByMapping

### Syntax

```
ReturnCode* logoutByMapping( char* argMapping,
                             MappingType argMappingType,
                             char* argDomain)
```

### Description

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

### Parameters

*argMapping*: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

*argMappingType*: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

*argDomain*: See description in *logoutByNameFromDomain* operation.

### Return Value

A pointer to a *ReturnCode* structure with a Boolean type:

- TRUE: if the subscriber was found and removed from the subscriber database.
- FALSE: if the subscriber was not found in the subscriber database.

### 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 List of Error Codes.

## Example

To remove IP address 192.168.12.5 from domain *subscribers*:

```
bapi.logoutByMapping(
    "192.168.12.5",
    IP_RANGE,
    "subscribers");
```

## loginCable

### Syntax

```
ReturnCode* loginCable(    char* argCpe,
                           char* argCm,
                           char* argIp,
                           int argLease,
                           char* argDomain,
                           char** argPropertyKeys,
                           char** argPropertyValues,
                           int argPropertySize)
```

### 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 to the SM. To log in a CM, specify the CM MAC address in both CPE and CM arguments. 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

`argCpe`: A unique identifier of the CPE (usually a MAC address).

`argCm`: A unique identifier of the cable modem (usually a MAC address).

`argIp`: the CPE IP address.

`argLease`: the CPE lease time.

`argDomain`: See explanation of *domains* ("[Subscriber Domains](#)" on page 2-9) in the *General API Concepts* 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*.

---

`argPropertyKeys`: See explanation of *property* ("[Subscriber Properties](#)" on page 2-10) keys and values in the *General API Concepts* 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.

`argPropertyValues`: See explanation of *property* ("[Subscriber Properties](#)" on page 2-10) keys and values in the *General API Concepts* chapter.

---

`argPropertySize`: The size of the `argPropertyKeys` and `argPropertyValues` arrays.

---

## Return Value

A pointer to a `ReturnCode` structure with a void type.

## Error Codes

None

## Examples

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

```
bapi.loginCable(
    "CM1",
    "CM1",
    "192.168.12.5",
    7200,                          // lease time in seconds
    "subscribers",
    NULL, NULL, 0);                // no properties
```

To add the IP address 192.168.12.50 to a CPE called *CPE1* behind *CM1* with a lease time of 1 hour:

```
bapi.loginCable(
    "CPE1",
    "CM1",
    "192.168.12.50",
    3600,                          // lease time in seconds
    "subscribers",
    NULL, NULL, 0);
```

## logoutCable

### Syntax

```
ReturnCode* logoutCable(char* argCpe,
                        char* argCm,
                        char* argIp,
                        char* argDomain)
```

### Description

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

### Parameters

*argCpe*: See description in the `loginCable` (on page 3-12) method.

*argCm*: See description in the `loginCable` (on page 3-12) method.

*argIp*: See description in the `loginCable` (on page 3-12) method.

*argDomain*: See description in the `loginCable` (on page 3-12) method.

### Return Value

A pointer to a `ReturnCode` structure with a Boolean type:

- TRUE: if the CPE was found and removed from the subscriber database.
- FALSE: if the CPE was not found in the subscriber database.

### Error Codes

None



## Examples

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

```
bool isExist = bapi.logoutCable(
    "CPE1",
    "CM1",
    "192.168.12.5",
    "subscribers");
```

## addSubscriber

### Syntax

```
ReturnCode* addSubscriber(    char* argName,
                             char** argMappings,
                             MappingType* argMappingTypes,
                             int argMappingsSize,
                             char** argPropertyKeys,
                             char** argPropertyValues,
                             int argPropertySize,
                             char** argCustomPropertyKeys,
                             char** argCustomPropertyValues,
                             int argCustomPropertySize,
                             char* argDomain)
```

### Description

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



**Note** It is recommended to call the `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 it, use `login`.

Example:

**Step 1** 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 an `login` operation with these NULL-value parameters will not change *AB*'s mappings; *AB* will be left with its previous IP mapping: *IP1*.

## Parameters

`argName`: See explanation of *subscriber name format* (on page 2-7) in the *General API Concepts* chapter.

`argMappings`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

`argMappingTypes`: See explanation of mappings and mapping types in the *General API Concepts* chapter.

`argMappingsSize`: The size of the `argMappings` and `argMappingTypes` arrays.

`argPropertyKeys`: See explanation of property keys and values in the *General API Concepts* chapter.

`argPropertyValues`: See explanation of property keys and values in the *General API Concepts* chapter.

`argPropertySize`: The size of the `argPropertyKeys` and `argPropertyValues` arrays.

`argCustomPropertyKeys`: See explanation of custom property keys and values in the *General API Concepts* chapter.

`argCustomPropertyValues`: See explanation of custom property keys and values in the *General API Concepts* chapter.

`argPropertySize`: the size of the `argCustomPropertyKeys` and `argCustomPropertyValues` arrays.

`argDomain`: See explanation of *domains* ("[Subscriber Domains](#)" on page 2-9) in the *General API Concepts* chapter.

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

## Return Value

A pointer to a `ReturnCode` structure with a void type.

## 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 indicates that invalid values were supplied for `propertyValues` parameter.

For error codes description see [List of Error Codes](#).

## Examples

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

```
char* propKeys[] = { "work_phone", "home_phone" };
char *propValues[] = { "123456", "898765" };

bapi.addSubscriber(
    "john",
    NULL, NULL, 0,           // dynamic mappings will be set by login
    NULL, NULL, 0,         // dynamic properties will be set by login
    propKeys, propValues, 2, // 2 custom properties
    "subscribers");        // default domain
```

## removeSubscriber

### Syntax

```
ReturnCode* removeSubscriber(char* argName)
```

### Description

Removes a subscriber completely from the SM database.

### Parameters

*argName*: See explanation of *Subscriber Name Format* (on page 2-7) in the *General API Concepts* chapter.

### Return Value

A pointer to a *ReturnCode* structure with a Boolean type:

- 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.

### 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 List of Error Codes.

### Example

To remove subscriber *john* entirely from the database:

```
bapi.removeSubscriber("john");
```

## removeAllSubscribers

### Syntax

```
ReturnCode* removeAllSubscribers()
```

### 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

A pointer to a `ReturnCode` structure with a void type.

### Error Codes

None.

## getNumberOfSubscribers

### Syntax

```
ReturnCode* getNumberOfSubscribers()
```

### Description

Retrieves the total number of subscribers in the SM database.

### Return Value

A pointer to a `ReturnCode` structure holding an integer describing the number of subscribers in the SM.

### Error Codes

None

## getNumberOfSubscribersInDomain

### Syntax

```
ReturnCode* getNumberOfSubscribersInDomain(char* argDomain)
```

### Description

Retrieves the number of subscribers in a subscriber domain.

## Parameters

`argDomain`: A name of a subscriber domain that exists in the SM's domain repository.

## Return Value

A pointer to a `ReturnCode` structure holding an integer describing the number of subscribers in the domain provided.

## 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 [List of Error Codes](#).

## getSubscriber

### Syntax

```
ReturnCode* getSubscriber(char* argName)
```

### 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 thrown.

### Parameters

`argName`: See explanation of *subscriber name format* (on page [2-7](#)) in the *General API Concepts* chapter.

### Return Value

A pointer to a `ReturnCode` structure holding An Array of `ReturnCode` structures with nine elements. The index values are listed in the following table. No array element is NULL.

Index 0	subscriber name (char*)
Index 1	array of mappings (char**)
Index 2	array of mapping types (short*)
Index 3	Domain name (char*)
Index 4	array of property names (char**)
Index 5	array of property values (char**)
Index 6	array of custom property names (char**)
Index 7	array of custom property values (char**)

Index 8            array of auto-logout time, as seconds from now, or value of **-1**  
if not set (long 1\*) one per mapping (index1)

## Error Codes

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

- ERROR\_CODE\_SUBSCRIBER\_DOES\_NOT\_EXIST

For error codes description see List of Error Codes.

## Example

To retrieve the subscriber record of *john*:

```
ReturnCode* sub = bapi.getSubscriber("john");
// sub name
char* name = sub->u.objectArray[0]->u.stringVal;

// sub mapping
char** mappings = sub->u.objectArray[1]->u.stringArrayVal;

// mappings types
short* types = sub->u.objectArray[2]->u.shortArrayVal;

char* domainName = (char*)sub->u.objectArray[3]->u.stringVal;

char** propertyNames = (char**)sub->u.objectArray[4]->u.stringArrayVal;

char** propertyValues = (char**)sub->u.objectArray[5]->u.stringArrayVal;

char** customPropertyName = (char**)sub->u.objectArray[6]->u.stringArrayVal;

char** customPropertyValues = (char**)sub->u.objectArray[7]-
>u.stringArrayVal;

long* autoLogoutTime = sub->u.objectArray[8]->u.longArrayVal;
```

## subscriberExists

### Syntax

```
ReturnCode* subscriberExists(char* argName)
```

### Description

Verifies that a subscriber exists in the SM database.

### Parameters

*argName*: See explanation of *subscriber name format* (on page [2-7](#)) in the *General API Concepts* chapter.

## Return Value

A pointer to a `ReturnCode` structure with a Boolean type:

- TRUE: if the subscriber was found in the SM database.
- FALSE: if the subscriber could not be found.

## Error Codes

None.

## subscriberLoggedIn

### Syntax

```
ReturnCode* subscriberLoggedIn(char* argName)
```

### 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 thrown.

### Parameters

`argName`: See explanation of *subscriber name format* (on page 2-7) in the *General API Concepts* chapter.

### Return Value

A pointer to a `ReturnCode` structure with a Boolean type:

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

### Error Codes

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

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see List of Error Codes.

## getSubscriberNameByMapping

### Syntax

```
ReturnCode* getSubscriberNameByMapping( char* argMapping,
                                        MappingType argMappingType,
                                        char* argDomain)
```

### Description

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

### Parameters

`argMapping`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

`argMappingType`: See explanation of mappings and *mapping types* ("[Network ID Mappings](#)" on page 2-8) in the *General API Concepts* chapter.

`argDomain`: 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

A pointer to a `ReturnCode` structure with a `String (char*)` type:

- Subscriber name: if a subscriber record was found.
- NULL: if no subscriber record with the supplied mapping could be found in the SM database.

### 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 List of Error Codes.

## getSubscriberNames

### Syntax

```
ReturnCode* getSubscriberNames( char* argFirstName,
                                int argAmount)
```



## Description

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

If `argFirstName` 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

`argFirstName`: Last subscriber name from last bulk (first name to look for). Use `NULL` to start with the first (alphabetic) subscriber.

`argAmount`: 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

A pointer to a `ReturnCode` structure with a String Array (`char**`) holding a list 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 `argAmount` and the SM limit (1000).

## Example

```

bool hasMoreSubscribers;
char* lastBulkEnd = NULL;
char tmpName[50];
int bulkSize = 100;

do
{
    ReturnCode* subscribers=

smApi.getSubscriberNames(lastBulkEnd,bulkSize);

    hasMoreSubscribers = false;

    if ((isReturnCodeError(subscribers) == false) &&
        (subscribers->type == STRING_ARRAY_T) &&
        (subscribers->u.stringArrayVal != NULL))
    {

        for (int i = 0; i < subscribers->size; i++)
        {
            // do something with subscribers->u.stringArrayVal[i]
        }

        if (subscribers->size == bulkSize)
        {
            hasMoreSubscribers = true;
            strcpy (tmpName, subscribers->u.stringArrayVal[bulkSize - 1]);
            lastBulkEnd = tmpName;
        }
    }
    freeReturnCode(subscribers);
} while (hasMoreSubscribers);

```

## getSubscriberNamesInDomain

### Syntax

```

ReturnCode* getSubscriberNamesInDomain( char* argFirstName,
                                       int argAmount,
                                       char* argDomain)

```

### Description

Gets subscribers from 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 [3-22](#)) operation.

## Parameters

`argFirstName`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argAmount`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argDomain`: 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 the documentation of the *Return Value* (on page 3-23) section of the `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation for more information.

## 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 List of Error Codes.

## getSubscriberNamesWithPrefix

### Syntax

```
ReturnCode* getSubscriberNamesWithPrefix(char* argFirstName,  
                                         int argAmount,  
                                         char* argPrefix)
```

### Description

Gets subscribers from the SM database whose names begin with a specified prefix.

The semantics of this operation are the same as the semantics of the `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

### Parameters

`argFirstName`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argAmount`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argPrefix`: 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* (on page 3-23) section of the `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

## Error Codes

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

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see List of Error Codes.

## getSubscriberNamesWithSuffix

### Syntax

```
ReturnCode* getSubscriberNamesWithSuffix(char* argFirstName,
                                         int argAmount,
                                         char* argSuffix)
```

### Description

Gets subscribers from 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 3-22) operation.

### Parameters

`argFirstName`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argAmount`: See description in `getSubscriberNames` ("[getSubscriberNames](#)" on page 3-22) operation.

`argSuffix` - 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* (on page 3-23) section of the `getSubscriberNames` operation.

### Error Codes

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

- `ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME`

For error codes description see List of Error Codes.

## getDomains

### Syntax

```
ReturnCode* getDomains()
```

### Description

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

### Return Value

A pointer to a `ReturnCode` structure with a String Array (`char**`) holding a complete list of subscriber domain names in the SM.

### Error Codes

None

## setPropertystoDefault

### Syntax

```
ReturnCode* setPropertiesToDefault(    char* argName,  
                                     char** argPropertyKeys,  
                                     int argPropertySize)
```

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, an `ERROR_CODE_ILLEGAL_STATE` error code is returned.

### Parameters

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

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

`argPropertySize`: The size of the `argPropertyKeys` array.

### Return Value

A pointer to a `ReturnCode` structure with a void type.

### 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 List of Error Codes.

## removeCustomProperties

### Syntax

```
ReturnCode* removeCustomProperties(    char* argName,
                                     char** argCustomPropertyKeys,
                                     int argCustomPropertySize)
```

### Description

Resets the specified custom properties of a subscriber.

### Parameters

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

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

`argCustomPropertySize`: the size of the `argCustomPropertyKeys` array.

### Return Value

A pointer to a `ReturnCode` structure with a void type.

### 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 *List of Error Codes* (on page A-1).

## C++ setLogger Method

### Syntax

```
void setLogger(Logger *argLogger)
```

### Description

Sets an implementation of the abstract `Logger` class. This method should be used for integrating the SM API log messages with the host application's log.

## Parameters

`argLogger`: an implementation of the abstract `Logger` class.

## Return Value

None.

## C++ `init` Method

### Syntax

```
Bool init( int argSupportedThreads,
          int argThreadPriority,
          Uint32 argBufferSize,
          Uint32 argKeepAliveDuration,
          Uint32 argConnectionTimeout)
```

### Description

Configures and initializes the API.



#### Note

This method must be called before performing any operation of the C++ API.

### Parameters

`argSupportedThreads`: The number of threads the API should support.

`argThreadPriority`: The priority for the PRPC protocol network thread.

`argBufferSize`: The internal buffer size (for default use 2000000 (2,000,000) bytes).

`argKeepAliveDuration`: A hint regarding the wanted delay between PRPC protocol keep-alive messages (default use 10 seconds).

`argConnectionTimeout`: A hint regarding the wanted timeout on a non-responding PRPC protocol connection (for default use 20 seconds).

### Return Value

Boolean value:

- TRUE: success
- FALSE: fail

### Example

```
SmBlockingApi bapi;
bool success = bapi.init(10,
                        0,
                        2000000, //default
                        10,      // default
```

## C SMB\_init Function

### Syntax

```
SMB_HANDLE SMB_init ( int argSupportedThreads,
                    int argThreadPriority,
                    Uint32 argBufferSize,
                    Uint32 argKeepAliveDuration,
                    Uint32 argConnectionTimeout)
```

### Description

Allocates, configures, and initializes the API.



#### Note

This method must be called before performing any operation of the C API.

### Parameters

`argSupportedThreads`: See description in *init* operation, *above* ("[C++ init Method](#)" on page 3-29).

`argThreadPriority`: See description in *init* operation, *above* ("[C++ init Method](#)" on page 3-29).

`argBufferSize`: See description in *init* operation, *above* ("[C++ init Method](#)" on page 3-29).

`argKeepAliveDuration`: See description in *init* operation, *above* ("[C++ init Method](#)" on page 3-29).

`argConnectionTimeout`: See description in *init* operation, *above* ("[C++ init Method](#)" on page 3-29).

### Return Value

SMB\_HANDLE handle to the API. If the handle equals NULL, the initialization failed. Otherwise a non-NULL value is returned.

### Example

```
SMB_HANDLE api;
// initialize an API
api = SMB_init(10, // 10 threads
              0,
              300000, // 3,000,000 bytes
              10,    // default
              30);  // 30 sec connection timeout
```



## C SMB\_release Function

### Syntax

```
void SMB_release(SMB_HANDLE argApiHandle)
```

### Description

Releases the resources used by the API. This function must be called at the end of the use of the API.

### Parameters

`argApiHandle`: the API handle received using the `SMB_init` function.

### Return Value

None.

## setName

### Syntax

```
void setName(char *argName)
```

### Description

Sets the name of the API, which serves as a unique identifier for the API-SM connection. The `setName` function should be called before calling the `connect` method.

### Parameters

`argName`: the API name.

### Return Value

None.

## connect

### Syntax

```
bool connect(char* argHostName, Uint16 argPort = 14374)
```

### Description

Attempts to establish a PRPC protocol connection to the SM.

## Parameters

`argHostName`: The SM IP-Address or hostname.

`argPort`: TCP port to connect the SM on (default is 14374).

## Return Value

Boolean value:

- TRUE: success
- FALSE: fail

## disconnect

### Syntax

```
bool disconnect()
```

### Description

Attempts to terminate the PRPC protocol connection to the SM.

### Return Value

Boolean value:

- TRUE: success
- FALSE: fail

## isConnected

### Syntax

```
bool isConnected();
```

### Description

Checks whether the PRPC protocol connection to the SM is up and running.

### Return Value

Boolean value:

- TRUE: connection is up.
- FALSE: connection is down.

## Blocking API C++ 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.

```
#include "SmApiBlocking.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    SmApiBlocking bapi;
    //initiation
    bapi.init();
    bapi.setReplyTimeout(300000); //set timeout for 5 minutes
    bapi.connect(argv[1]);        // connect to the SM
    //operations
    ReturnCode* domains = bapi.getDomains();
    ReturnCode* totalSubscribers=bapi.getNumberOfSubscribers();
    if ((isReturnCodeError(domains) == false) &&
        (isReturnCodeError(totalSubscribers) == false))
    {
        printf(
            "number of susbcribers in the database:\t\t %d\n",
            totalSubscribers->u.intVal);
        for (int i=0; i<domains->size; i++)
        {
            ReturnCode* numberOfSubscribersInDomain=
            bapi.getNumberOfSubscribersInDomain(
                domains->u.stringArrayVal[i]);
            if (isReturnCodeError(numberOfSubscribersInDomain)
                ==
false)
            {
                printf(
                    "number of susbcribers domain %s:\t\t%d\n",
                    domains->u.stringArrayVal[i],
                    numberOfSubscribersInDomain->u.intVal);
            }
            freeReturnCode (numberOfSubscribersInDomain);
        }
    }
    freeReturnCode (domains);
    freeReturnCode (totalSubscribers);
    //finalization
    bapi.disconnect();
    return 0;
}
```

## 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.

```

#include "SmApiBlocking.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    checkArguments(argc, argv);
    SmApiBlocking bapi;
    //initiation
    bapi.init();
    bapi.setReplyTimeout(10000);           //set timeout for 10 seconds
    bapi.connect(argv[1]);                 // connect to the SM
    //add subscriber
    printf("adding subscriber to SM\n");
    MappingType type = IP_RANGE;
    char* customKey = "custom-key";
    char* customVal = "custom-value";
    ReturnCode* ret = bapi.addSubscriber(
        argv[2],           // name
        &(argv[3]),       // mapping`
        &type,             // mapping type
        1,                 // one mapping
        &(argv[4]),       // property key
        &(argv[5]),       // property value
        1,                 // number of properties
        &customKey,       //custom property key
        &customVal,       //custom property value
        1,                 // number of custom properties
        argv[6]);         //domain

    freeReturnCode (ret);

    //Print subscriber
    printf("Printing subscriber:\n");
    ReturnCode* subfields = bapi.getSubscriber(argv[1]);
    if (isReturnCodeError(subfields) == false)
    {
        printf("\tname:\t\t%s\n",
            subfields->u.objectArray[0]-
>u.stringVal);
        printf("\tmapping:\t\t%s\n",
            subfields->u.objectArray[1]->u.stringArrayVal[0]);
        printf("\tdomain:\t\t%s\n",
            subfields-
>u.objectArray[3]->u.stringVal);
        printf("\tautologout:\t%d\n",
            subfields-
>u.objectArray[8]->u.intVal);
        // Remove subscriber
        printf("removing subscriber from SM\n");
        bapi.removeSubscriber(argv[1]);
    }
    else
    {
        printf("error in subscriber retrieval\n");
    }
    freeReturnCode(subfields);
    //finalization
    bapi.disconnect();
    return 0;
}

void checkArguments(int argc, char* argv[])

```

```
{
if (argc != 7)
{
printf("usage: AddPrintRemove <SM-address> <subscriber-name> "
"<IP mapping> <property-key> <property-value> <domain>");
exit(1);
}
}
```

## Blocking API C Code Examples

This section provides 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.

```

#include "SmApiBlocking_c.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    //initiation
    SMB_HANDLE bapi = SMB_init(10,0,2000000,10,20);
    if (bapi == NULL)
    {
        // init failure
        return -1;
    }
    SMB_setReplyTimeout(bapi,300000); //set timeout for 5 minutes
    SMB_connect(bapi,argv[1],14374); // connect to the SM
    //operations
    ReturnCode* domains = SMB_getDomains(bapi);
    ReturnCode* totalSubscribers= SMB_getNumberOfSubscribers(bapi);
    if ((isReturnCodeError(domains) == false) &&
        (isReturnCodeError(totalSubscribers) == false))
    {
        printf("number of susbcribers in the database:\t\t %d\n",
            totalSubscribers->u.intVal);
        for (int i=0; i<domains->size; i++)
        {
            ReturnCode* numberOfSubscribersInDomain=
                SMB_getNumberOfSubscribersInDomain(bapi,
                    domains-
>u.stringArrayVal[i]);
            if(isReturnCodeError(numberOfSubscribersInDomain) ==
                false)
            {
                printf(
                    "number of susbcribers domain
%s:\t\t%d\n",
                    domains->u.stringArrayVal[i],
                    numberOfSubscribersInDomain-
>u.intVal);
                freeReturnCode (numberOfSubscribersInDomain);
            }
        }
        freeReturnCode (domains);
        freeReturnCode (totalSubscribers);
    }
    //finalization
    SMB_disconnect(bapi);
    SMB_release(bapi);
    return 0;
}

```

## 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.

```
#include "SmApiBlocking_c.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    checkArguments(argc,argv);

    //initiation
    SMB_HANDLE bapi = SMB_init(10,0,2000000,10,20);
    if (bapi == NULL)
    {
        // init failure
        return -1;
    }
    SMB_setReplyTimeout(bapi,10000); //set timeout for 10 seconds
    SMB_connect(bapi,argv[1], 14374); // connect to the SM
    //add subscriber
    printf("adding subscriber to SM\n");
    MappingType type = IP_RANGE;
    char* customKey = "custom-key";
    char* customVal = "custom-value";
    ReturnCode* ret = SMB_addSubscriber(
        bapi, // handle
        argv[2], // name
        &(argv[3]), // mapping`
        &type, // mapping type
        1, // one mapping
        &(argv[4]), // property key
        &(argv[5]), // property value
        1, // number of properties
        &customKey, //custom property key
        &customVal, //custom property value
        1, // number of custom properties
        argv[6]); //domain

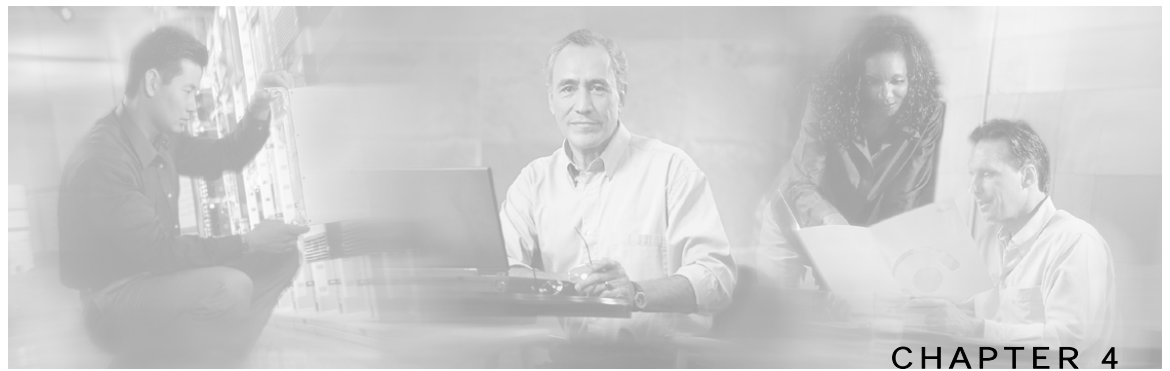
    freeReturnCode (ret);

    //Print subscriber
    printf("Printing subscriber:\n");
    ReturnCode* subfields = SMB_getSubscriber(bapi,argv[2]);
    if (isReturnCodeError(subfields) == false)
    {
        printf("\tname:\t\t%s\n",
            subfields->u.objectArray[0]-
>u.stringVal);
        printf("\tmapping:\t\t%s\n",
            subfields->u.objectArray[1]->u.stringArrayVal[0]);
        printf("\tdomain:\t\t\t%s\n",
            subfields-
>u.objectArray[3]->u.stringVal);
        printf("\tautologout:\t\t%d\n",
            subfields-
>u.objectArray[8]->u.intVal);
        // Remove subscriber
        printf("removing subscriber from SM\n");
        SMB_removeSubscriber(bapi,argv[2]);
    }
}
```

```
    }
    else
    {
        printf("error in subscriber retrieval\n");
    }
    freeReturnCode(subfields);
    //finalization
    SMB_disconnect(bapi);
    SMB_release(bapi);
    return 0;
}

void checkArguments(int argc, char* argv[])
{
    if (argc != 7)
    {
        printf(
            "usage: AddPrintRemove <SM-address> <subscriber-name> "
            "<IP mapping> <property-key> <property-value> <domain>");
        exit(1);
    }
}
```





# Non-blocking API

---

This chapter introduces the Result Handler Callbacks, a feature unique to the Non-blocking API. It also lists all methods of the Non-blocking API, and ends with some code examples.

This chapter contains the following sections:

- [Multi-threading Support](#) 4-1
- [Result Handler Callbacks](#) 4-2
- [Non-blocking API Methods](#) 4-4
- [Non-blocking API C++ Code Examples](#) 4-7
- [Non-blocking API C Code Examples](#) 4-10

## 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.

---

## Result Handler Callbacks

The Non-blocking API enables setting result handler callbacks. The result handler callbacks are two functions for `handleSuccess` and `handleError`, as outlined in the following code.

```
/* operation failure callback specification */
typedef void (*OperationFailCallbackFunc)(Uint32 argHandle,
                                           ReturnValue
                                           *argReturnCode);

/* operation success callback specification */
typedef void (*OperationSuccessCallbackFunc)(Uint32 argHandle,
                                              ReturnValue *argReturnCode);
```

You should implement these callbacks 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.

Both `handleSuccess` and `handleError` callbacks accept two parameters:

- **Handle:** Each API operation's return-value is a handle of type `Uint32`. 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 returned as a pointer of type `ResultCode`.

Example:

- The following example is a simple implementation of a result handler that counts the number of success/failure operations. 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.

```
#include "GeneralDefs.h"
#include "SmApiNonBlocking.h"
```

- `#include <stdio.h>`

```
int successCnt = 0;
int failCnt = 0;

void onOperationFail(UINT32 argHandle,
ReturnCode* argReturnCode)
{
    failCnt++;
    if (argReturnCode != NULL)
    {
        freeReturnCode(argReturnCode);
    }
}

void onOperationSuccess(UINT32 argHandle,
ReturnCode* argReturnCode)
{
    successCnt++;

    if (argReturnCode != NULL)
    {
        freeReturnCode(argReturnCode);
    }
}

int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("usage: ResultHandlerExample <sm-ip>");
        exit(1);
    }

    //note the order of operations!
    SmApiNonBlocking nbapi;
    nbapi.init();
    nbapi.connect(argv[1]);
    nbapi.setReplyFailCallBack(onOperationFail);
    nbapi.setReplySuccessCallBack(onOperationSuccess);
    nbapi.login(...);
    ...

    nbapi.disconnect();

    return 0;
}
```

## Non-blocking API Methods

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

Some of the methods return a non-negative `int` handle that may be used to correlate operation calls and their results (see the *Result Handler Callbacks* (on page 4-2) section). If an internal error had occurred a negative value is returned and the operation is not performed.

The operation results passed to the result handler callbacks are the same as the return values described in the same method in the *Blocking API* (on page 3-1), except that: return values of `Void` are translated to `NULL`.



### Note

The error/fail callback will be handed with an error **if and only if** the matching operation in the Blocking API would return an error code with the same arguments according to the SM database state at the time of the call.

The C and C++ API share the same function signature, except for an `SMNB_` prefix for all Non-blocking C APIs function names, and an API handle of type `SMNB_HANDLE` as the first parameter in all functions. Other differences between the APIs are explained in the function description.

The following methods are described:

- *login* (on page 4-5)
- *logoutByName* (on page 4-5)
- *logoutByNameFromDomain* (on page 4-5)
- *logoutByMapping* (on page 4-5)
- *loginCable* (on page 4-6)
- *logoutCable* (on page 4-6)
- C++ *setLogger Method* (on page 4-6)
- C++ *init Method* (on page 4-6)
- C *SMNB\_init Function* (on page 4-6)
- C *SMNB\_release Function* (on page 4-7)
- *setName* (on page 4-7)
- *connect* (on page 4-7)
- *disconnect* (on page 4-7)
- *isConnected* (on page 4-7)

## login

### Syntax

```
int login( char* argName,
          char** argMappings,
          MappingType* argMappingTypes,
          int argMappingsSize,
          char** argPropertyKeys,
          char** argPropertyValues,
          int argPropertySize,
          char* argDomain,
          bool argIsAdditive,
          int argAutoLogoutTime)
```

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

## logoutByName

### Syntax

```
int logoutByName(char* argName,
                 char** argMappings,
                 MappingType* argMappingTypes,
                 int argMappingsSize)
```

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

## logoutByNameFromDomain

### Syntax

```
int logoutByNameFromDomain (char* argName,
                             char** argMappings,
                             MappingType* argMappingTypes,
                             int argMappingsSize,
                             char* argDomain)
```

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

## logoutByMapping

### Syntax

```
int logoutByMapping( char* argMapping,
                     MappingType argMappingType,
                     char* argDomain)
```

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

## loginCable

### Syntax

```
int loginCable( char* argCpe,
               char* argCm,
               char* argIp,
               int argLease,
               char* argDomain,
               char** argPropertyKeys,
               char** argPropertyValues,
               int argPropertySize)
```

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

## logoutCable

### Syntax

```
int logoutCable( char* argCpe,
                 char* argCm,
                 char* argIp,
                 char* argDomain)
```

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

## C++ setLogger Method

### Syntax

```
void setLogger(Logger *argLogger)
```

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

## C++ init Method

### Syntax

```
Bool init( int argThreadPriority = 0,
           Uint32 argBufferSize = DEFAULT_BUFFER_SIZE,
           Uint32 argKeepAliveDuration = DEFAULT_KEEP_ALIVE_DURATION,
           Uint32 argConnectionTimeout= DEFAULT_CONNECTION_TIMEOUT)
```

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

## C SMNB\_init Function

### Syntax

```
SMNB_HANDLE SMNB_init( int argThreadPriority,
                       Uint32 argBufferSize,
                       Uint32 argKeepAliveDuration,
                       Uint32 argConnectionTimeout)
```

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

## Return Value

SMNB\_HANDLE handle to the API. If the handle equals NULL, the initialization failed. Otherwise a non-NULL value is returned.

## C SMNB\_release Function

### Syntax

```
void SMNB_release(SMNB_HANDLE argApiHandle)
```

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

## setName

### Syntax

```
void setName(char *argName)
```

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

## connect

### Syntax

```
bool connect(char* argHostName, Uint16 argPort = 14374)
```

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

## disconnect

### Syntax

```
bool disconnect()
```

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

## isConnected

### Syntax

```
bool isConnected();
```

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

## Non-blocking API C++ 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*.

```
#include "SmApiNonBlocking.h"
#include <stdio.h>

void connectionIsDown()
{
    printf("disconnect listener callback:: connection is down\n");
}

int count = 0;

//prints every error that occurs
void handleError(UINT32 argHandle, ReturnCode* argReturnCode)
{
    ++count;
    printf("\terror %d:\n",count);
    printReturnCode(argReturnCode);
    freeReturnCode(argReturnCode);
}

//prints a success result every 100 results
void handleSuccess(UINT32 argHandle, ReturnCode* argReturnCode)
{
    if (++count%100 == 0)
    {
        printf("\tresult %d:\n",count);
        printReturnCode(argReturnCode);
    }

    freeReturnCode(argReturnCode);
}

//waits for result number 'last result' to arrive
void waitForLastResult(int lastResult)
{
    while (count<lastResult)
    {
        ::Sleep(100);
    }
}

void checkTheArguments(int argc, char* argv[])
{
    if (argc != 4)
    {
        printf
            ("usage: LoginLogout <SM-address> <domain> <num-
            subscribers>");
        exit(1);
    }
}

void main (int argc, char* argv[])
{
    //check arguments
    checkTheArguments(argc, argv);
    int numSubscribersToLogin = atoi(argv[3]);
}
```



```

//instantiation
SmApiNonBlocking nbapi;
//initiation
nbapi.init();
nbapi.setDisconnectListener(connectionIsDown);
nbapi.connect(argv[1]);
nbapi.setReplyFailCallBack(handleError);
nbapi.setReplySuccessCallBack(handleSuccess);
//login
char name[10];
char ipString[15];
char* ip = &(ipString[0]);
MappingType type = IP_RANGE;
Uint32 ipVal = 0x0a000000;

printf("login of %d subscribers\n",numSubscribersToLogin);
for (int i=0; i<numSubscribersToLogin; i++)
{
    sprintf((char*)name,"s%d",i);
    sprintf((char*)ip,"%d.%d.%d.%d",
        (int)((ipVal & 0xFF000000) >> 24),
        (int)((ipVal & 0x00FF0000) >> 16),
        (int)((ipVal & 0x0000FF00) >> 8),
        (int)(ipVal & 0x000000FF));
    ipVal++;

    nbapi.login(name, //subscriber name
               &ip, //a single ip mapping
               &type,
               1,
               NULL, //no properties
               NULL,
               0,
               argv[2], //domain
               false, //mappings are not additive
               -1); //disable auto-logout
}
waitForLastResult(numSubscribersToLogin);
//logout
printf("logout of %d subscribers",numSubscribersToLogin);
ipVal = 0x0a000000;
for (i=0; i<numSubscribersToLogin; i++)
{
    sprintf((char*)ip,"%d.%d.%d.%d",
        (int)((ipVal & 0xFF000000) >> 24),
        (int)((ipVal & 0x00FF0000) >> 16),
        (int)((ipVal & 0x0000FF00) >> 8),
        (int)(ipVal & 0x000000FF));
    ipVal++;

    nbapi.logoutByMapping(ip,
                          type,
                          argv[2]);
}
waitForLastResult(numSubscribersToLogin*2);
nbapi.disconnect();
}

```

## Non-blocking API C 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*.

```

#include "SmApiNonBlocking_c.h"
#include <stdio.h>

void connectionIsDown()
{
    printf("disconnect listener callback:: connection is down\n");
}

int count = 0;

//prints every error that occurs
void handleError(UINT32 argHandle, ReturnCode* argReturnCode)
{
    ++count;
    printf("\terror %d:\n",count);
    printReturnCode(argReturnCode);
    freeReturnCode(argReturnCode);
}

//prints a success result every 100 results
void handleSuccess(UINT32 argHandle, ReturnCode* argReturnCode)
{
    if (++count%100 == 0)
    {
        printf("\tresult %d:\n",count);
        printReturnCode(argReturnCode);
    }

    freeReturnCode(argReturnCode);
}

//waits for result number 'last result' to arrive
void waitForLastResult(int lastResult)
{
    while (count<lastResult)
    {
        ::Sleep(100);
    }
}

void checkTheArguments(int argc, char* argv[])
{
    if (argc != 3)
    {
        printf
            ("usage: LoginLogout <SM-address> <domain> <num-susscribers>");
        exit(1);
    }
}

void main (int argc, char* argv[])
{
    //check arguments
    checkTheArguments(argc, argv);
    int numSubscribersToLogin = atoi(argv[3]);

    //instantiation
    SMNB_HANDLE nbapi = SMNB_init(0,2000000,10,30);
    if (nbapi == NULL)
    {
        exit(1);
    }
}

```

```

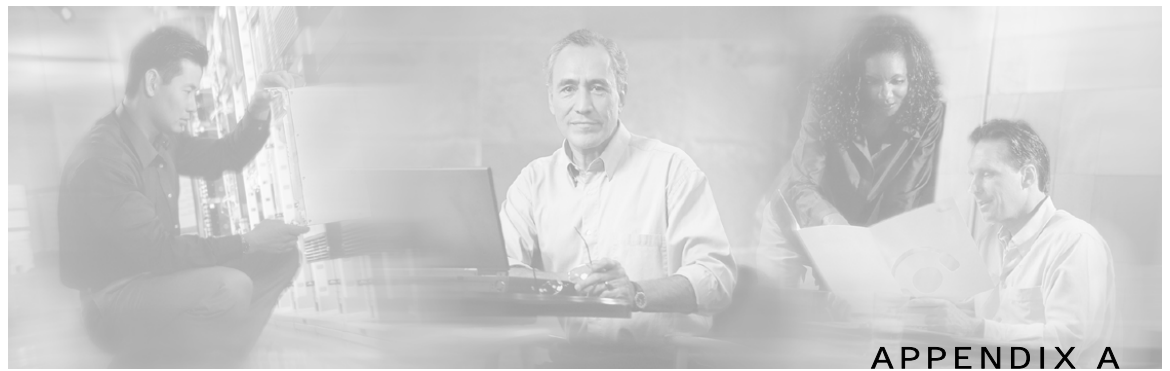
SMNB_setDisconnectListener(nbapi,connectionIsDown);
SMNB_connect(nbapi,argv[1],14374);
SMNB_setReplyFailCallBack(nbapi,handleError);
SMNB_setReplySuccessCallBack(nbapi,handleSuccess);
//login
char name[10];
char ipString[15];
char* ip = &(ipString[0]);
MappingType type = IP_RANGE;
Uint32 ipVal = 0x0a000000;

printf("login of %d subscribers\n",numSubscribersToLogin);
for (int i=0; i<numSubscribersToLogin; i++)
{
    sprintf((char*)name,"s%d",i);
    sprintf((char*)ip,"%d.%d.%d.%d",
            (int)((ipVal & 0xFF000000) >> 24),
            (int)((ipVal & 0x00FF0000) >> 16),
            (int)((ipVal & 0x0000FF00) >> 8),
            (int)(ipVal & 0x000000FF));
    ipVal++;

    SMNB_login( nbapi,
                name,           //subscriber name
                &ip,           //a single ip mapping
                &type,
                1,
                NULL,           //no properties
                NULL,
                0,
                argv[2],        //domain
                false,         //mappings are not additive
                -1);           //disable auto-logout
}
waitForLastResult(numSubscribersToLogin);
//logout
printf("logout of %d subscribers",numSubscribersToLogin);
ipVal = 0x0a000000;
for (i=0; i<numSubscribersToLogin; i++)
{
    sprintf((char*)ip,"%d.%d.%d.%d",
            (int)((ipVal & 0xFF000000) >> 24),
            (int)((ipVal & 0x00FF0000) >> 16),
            (int)((ipVal & 0x0000FF00) >> 8),
            (int)(ipVal & 0x000000FF));
    ipVal++;

    SMNB_logoutByMapping(nbapi,
                          ip,
                          type,
                          argv[1]);
}
waitForLastResult(numSubscribersToLogin*2);
SMNB_disconnect(nbapi);
SMNB_release(nbapi);
}

```



## List of Error Codes

Error codes are used for interpreting the actual error for which a `ReturnCode` (holding an `ErrorCode`) was returned.

The error code enumeration is given in the `GeneralDefs.h` header file. 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.

<b>Error Code</b>	<b>Description</b>
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.
ERROR_CODE_CLIENT_OPERATION_TIMEOUT	Blocking API operation result did not return till the reply timeout expired.



# Index

---

## A

- Adding Subscriber, Printing Information, Removing Subscriber • 3-33, 3-37
- addSubscriber • 3-15
- API Construction • 2-4
- API Finalization • 2-5
- API Initialization • 2-3
- Audience • v

## B

- Blocking API • 2-2, 3-1
  - code examples • 3-32, 3-35
- Blocking API C Code Examples • 3-35
- Blocking API C++ Code Examples • 3-32
- Blocking API Example • 2-3
- Blocking API Methods • 3-3
- Blocking API setup • 2-5
- Blocking API/Non-blocking API • 2-1

## C

- C SMB\_init Function • 3-30
- C SMB\_release Function • 3-31
- C SMNB\_init Function • 4-6
- C SMNB\_release Function • 4-7
- C vs. C++ API • 2-2
- C++ init Method • 3-29, 4-6
- C++ setLogger Method • 3-28, 4-6
- Cisco TAC Website • vi
- Code examples
  - blocking API • 3-32, 3-35
  - non-blocking API • 4-7, 4-10
- Compiling and Running • 1-3
- connect • 3-31, 4-7
- Connecting to the SM • 2-5
- Custom Properties • 2-10

## D

- Definition • 2-7
- Definitions • 2-6
- Description • 3-5, 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, 3-27, 3-28, 3-29, 3-30, 3-31, 3-32
- disconnect • 3-32, 4-7
- Disconnect Callback Listener • 2-11
- Document Conventions • v

## E

- Error Code Structure • 2-7
- Error Codes • 3-6, 3-9, 3-10, 3-11, 3-13, 3-14, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-25, 3-26, 3-27, 3-28
- Example • 2-3, 2-7, 3-7, 3-9, 3-11, 3-12, 3-17, 3-20, 3-24, 3-29, 3-30
- Examples • 3-14, 3-15, 3-17
- Extracting the Package • 1-1

## G

- General API Concepts • 2-1
- getDomains • 3-27
- getNumberOfSubscribers • 3-18
- getNumberOfSubscribersInDomain • 3-18
- getSubscriber • 3-19
- getSubscriberNameByMapping • 3-22
- getSubscriberNames • 3-22
- getSubscriberNamesInDomain • 3-24
- getSubscriberNamesWithPrefix • 3-25
- getSubscriberNamesWithSuffix • 3-26
- Getting Number of Subscribers • 3-33, 3-35
- Getting Started • 1-1

## H

- Handle Pointers • 2-3

**I**

Installation • 1-1  
isConnected • 3-32, 4-7

**L**

Linux (Red Hat) • 1-4  
List of Error Codes • A-1  
Logging Capabilities • 2-10  
login • 3-5, 4-5  
Login and Logout • 4-8, 4-10  
login method  
    non-blocking API • 4-5  
loginCable • 3-12, 4-6  
loginCable method  
    non-blocking API • 4-6  
logoutByMapping • 3-11, 4-5  
logoutByMapping method  
    non-blocking API • 4-5  
logoutByName • 3-8, 4-5  
logoutByName method  
    non-blocking API • 4-5  
logoutByNameFromDomain • 3-9, 4-5  
logoutByNameFromDomain method  
    non-blocking API • 4-5  
logoutCable • 3-14, 4-6  
logoutCable method  
    non-blocking API • 4-6

**M**

Method Names • 2-2  
Multi-threading Support • 3-1, 4-1

**N**

Network ID Mappings • 2-8  
Non-blocking API • 2-2, 4-1  
    code examples • 4-7, 4-10  
    login • 4-5  
    loginCable • 4-6  
    logoutByMapping • 4-5  
    logoutByName • 4-5  
    logoutByNameFromDomain • 4-5  
    logoutCable • 4-6  
Non-blocking API C Code Examples • 4-10  
Non-blocking API C++ Code Examples • 4-7  
Non-Blocking API example • 2-3  
Non-blocking API Methods • 4-4  
Non-blocking API setup • 2-5

**O**

Opening a TAC Case • vi  
Operation Timeout Error Code • 3-2

**P**

Package Content • 1-2  
Parameters • 3-5, 3-8, 3-10, 3-11, 3-13, 3-14, 3-16, 3-17, 3-19, 3-20, 3-21, 3-22, 3-23, 3-25, 3-26, 3-27, 3-28, 3-29, 3-30, 3-31, 3-32  
Platforms and Compilers • 1-1  
Preface • v  
Purpose • v

**R**

Related Publications • v  
Reliability • 2-2  
removeAllSubscribers • 3-18  
removeCustomProperties • 3-28  
removeSubscriber • 3-17  
Result Handler Callbacks • 4-2  
Return Code Structure • 2-5  
Return Value • 3-6, 3-8, 3-10, 3-11, 3-13, 3-14, 3-16, 3-17, 3-18, 3-19, 3-21, 3-22, 3-23, 3-25, 3-26, 3-27, 3-28, 3-29, 3-30, 3-31, 3-32, 4-7

**S**

setName • 3-31, 4-7  
setPropertyToDefault • 3-27  
SETTING the LEG NAME • 2-4  
Setup Operations • 2-5  
Signal Handling • 2-11  
SM Setup • 1-4  
Solaris • 1-3  
Specifying IP Address Mapping • 2-8  
Specifying IP Range Mapping • 2-9  
Specifying VLAN Tag Mapping • 2-9  
Subscriber Domains • 2-9  
Subscriber Name Format • 2-7  
Subscriber Properties • 2-10  
subscriberExists • 3-20  
subscriberLoggedIn • 3-21  
Syntax • 3-5, 3-8, 3-9, 3-11, 3-12, 3-14, 3-15, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-24, 3-25, 3-26, 3-27, 3-28, 3-29, 3-30, 3-31, 3-32, 4-5, 4-6, 4-7



**T**

TAC Case Priority Definitions • vii

Technical Support • vi

**W**

Windows • 1-3