Cisco SCMS SM C/C++ API Programmer Guide

Release 3.1

May 2007

Americas 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: 408 527-0883

Customer Order Number:
Text Part Number: OL-7203-05

CONTENTS

About This Guide

Revised: May 30, 2007, OL-7203-05

This document explains the SCMS SM C/C++ API and how to install, compile, and run the API.

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

- 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.

Note A set of APIs with exactly the same functionality is also available for the Java environment.

This introduction provides information about the following topics:

- Audience
- Document History
- Organization
- Document Conventions
- Related Documentation
- Obtaining Documentation, Obtaining Support, and Security Guidelines

Audience

This guide is for the networking or computer technician responsible for configuring the Subscriber Manager. It is also intended for the operator who manages the SCE platforms.

Document History

Cisco Service Control Release	Part Number	Publication Date
Release 3.1.0	OL-7203-05	May, 2007

Description of Changes

- Support for moving subscribers between domains. See Subscriber Domains and the Parameters section of the login method.
- Updated Subscriber Name Format.

Cisco Service Control Release	Part Number	Publication Date
Release 3.0.5	OL-7203-04	November, 2006

Description of Changes

- Addition of the reconnect timeout parameter to the C++ init method. See C++ init Method.
- Addition of the setReconnectTimeout method. See setReconnectTimeout.

Cisco Service Control Release	Part Number	Publication Date
Release 3.0.3	OL-7203-03	May, 2006

Description of Changes

- Updated documentation for release 3.0.3.

Cisco Service Control Release	Part Number	Publication Date
Release 3.0	OL-7203-02	December, 2005

Description of Changes

- Reorganization of documentation. No major changes or new features were added to this revision.

Cisco Service Control Release	Part Number	Publication Date
Release 2.5.7	OL-7203-01	May, 2005

Description of Changes

- This is the first version of this document.

Organization

The major sections of this guide are as follows:

Table 1 How This Guide Is Organized

Chapter	Title	Description
Chapter 1	Getting Started	Describes the platforms on which the C/C++ API can be used, how to install, compile, and start running the C/C++ API component.
Chapter 2	General API Concepts	Provides a description of the various concepts that are used when working with the SM C/C++ API.
Chapter 3	Blocking API	Describes the features and operations of the Blocking API and provides code examples.
Chapter 4	Non-blocking API	Describes the features and operations of the Non-blocking API and provides code examples.
Appendix A	List of Error Codes	Provides a list of error codes that are used in the C/C++ API.

Document Conventions

This guide uses the following conventions:

- **Bold** is used for commands, keywords, and buttons.
- *Italics* are used for command input for which you supply values.
- `Screen` font is used for examples of information that are displayed on the screen.
- **`Bold screen`** font is used for examples of information that you enter.
- Vertical bars (|) indicate separate alternative, mutually exclusive elements.
- Square brackets ([ ]) indicate optional elements.
- Braces ({}) indicate a required choice.
- Braces within square brackets ([{}]) indicate a required choice within an optional element.

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the guide.

Timesaver Means the *described action saves time*. You can save time by performing the action described in the paragraph.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Warning **Means *danger*. You are in a situation that could cause bodily injury. Before you work on any equipment, you must be aware of the hazards involved with electrical circuitry and familiar with standard practices for preventing accidents. To see translated versions of warnings, refer to the *Regulatory Compliance and Safety Information* document that accompanied the device.**

Related Documentation

Use this Cisco SCMS SM C/C++ API Programmer Guide in conjunction with all of the SCMS Subscriber Manager User, API, and Reference Guides.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

CHAPTER 1

Getting Started

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

- Platforms and Compilers
- Installation Package Contents
- How to Install the SCMS SM C/C++ API
- How to Compile and Run the SCMS SM C/C++ API

Platforms and Compilers

The SCMS 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 Package Contents

For brevity, the installation directory **sm-c-api-vvv.bb** is referred to as **<installdir>**, where **vvv** and **bb** stand for the SCMS SM C/C++ API version and build number.

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 the C++ API internal header files.

Table 1-1 Layout of Installation Directory

Folder	Subfolder (as applicable)	File Name
<installdir>		README.csmapi
	include	BasicTypes.h
		Common.h
		GeneralDefs.h

Table 1-1 Layout of Installation Directory

Folder	Subfolder (as applicable)	File Name
		Logger.h
		PrintLogger.h
		SmApiBlocking.h
		SmApiBlocking_c.h
		SmApiNonBlocking.h
		SmApiNonBlocking_c.h
	include/system	OperationHandleInterface.h
		OperationResultInterface.h
	linux	libsmapi.so
	solaris	libsmapi.so
	winnt	asn1ber.dll
		asn1ber.lib
		asn1rt.dll
		asn1rt.lib
		SmApi.dll
		SmApi.lib

How to Install the SCMS SM C/C++ API

The SCMS SM C/C++ API distribution is part of the SCMS SM-LEG distribution file and is located in the sm_api directory. The SCMS SM C/C++ API is packaged in a UNIX tar file. You can extract the SCMS SM C/C++ API using the UNIX tar utility or most Windows compression utilities.

- Installing the Distribution on the Unix and Linux Platforms
- Installing the Distribution on Windows

Installing the Distribution on the Unix and Linux Platforms

1. Extract the SCMS SM-LEG distribution file.
2. Locate the C/C++ SM API distribution file **sm-c-api-dist.tar**
3. Extract the C/C++ API package tar:

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

Installing the Distribution on Windows

1. Extract the package using a zip extractor (such as WinZip).
2. Respond as appropriate to the prompts from the installation wizard.

How to Compile and Run the SCMS SM C/C++ API

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 on the SM must be started.

The PRPC server is a proprietary RPC protocol designed by Cisco. For additional information, see the *Cisco SCMS Subscriber Manager User Guide*.

The following sections describe how to compile and run the API on the supported software platforms:

- Compiling and Running a Program on Windows
- Compiling and Running a Program on Solaris
- Compiling and Running a Program on Red Hat Linux

Compiling and Running a Program on Windows

SUMMARY STEPS

1. Ensure that **smapi.dll** and the other DLL files are in your path or in the directory of your executable.
2. Ensure that the **include** folder is in your include path of the compilation.
3. Ensure that the smapi.lib file is in your linkage path.
4. Include the relevant API header file in your source code, and compile your code.

DETAILED STEPS

Step 1 Ensure that **smapi.dll** and the other DLL files are in your path or in the directory of your executable.

Step 2 Ensure that the **include** folder is in your include path of the compilation.

Example using Microsoft Visual C++ 6:

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

Step 3 Ensure that the smapi.lib file is in your linkage path.

Example using Microsoft Visual C++ 6:

Enter the project settings and click 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.

Compiling and Running a Program on Solaris

SUMMARY STEPS

1. Ensure that **libsmapi.so** is in your LD_LIBRARY_PATH.
2. Ensure that the **include** folder is in your include path of the compilation.

3. Ensure that the **libsmapi.so** file is in your linkage line or load it dynamically.

DETAILED STEPS

Step 1 Ensure that **libsmapi.so** is 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 Ensure that the **include** folder is 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 Ensure that the **libsmapi.so** file is 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
```

Compiling and Running a Program on Red Hat Linux

SUMMARY STEPS

1. Ensure that **libsmapi.so** is in your LD_LIBRARY_PATH.
2. Ensure that the **include** folder is in your include path of the compilation.
3. Ensure that the **libsmapi.so** file is in your linkage line or load it dynamically.

DETAILED STEPS

Step 1 Ensure that **libsmapi.so** is in your LD_LIBRARY_PATH.

For example, when using the **Bash** shell type, use the following command line, entering your password if prompted:

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

Step 2 Ensure that the **include** folder is 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-headerfile-folder/system/ TestSmApi.cpp
```

Step 3 Ensure that the **libsmapi.so** file is in your linkage line or load it dynamically.

Specify the location of the **libsmapi.so** file 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<libpath>
```

CHAPTER 2

General API Concepts

This module provides a description of the various concepts that are used when working with the SM C/C++ API.

Information About the Blocking API/Non-blocking API

This topic describes the differences between the blocking API and non-blocking API operations.

- Blocking API
- Non-blocking API

Blocking API

In a blocking API operation, which is the most common, 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 operation, every method returns *immediately* , even before the completion of its operation. The operation results are returned either 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 calling program to continue performing other tasks and it 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 SCMS SM C/C++ API is reliable in the sense that the operations performed on the API are kept until the SM returns their associated results. If the connection to the SM is lost, operations that were not sent to the SM and operations whose results did not yet return from the SM are sent to the SM immediately on reconnection. The order of operation invocation is maintained at all times.

C versus 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.

The following sections describe the C/C++ API differences and provide some examples.

- Method Names
- Handle Pointers

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:

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

Note C++ API methods are documented in this guide using their name and signature.

Example:

Both the blocking and non-blocking C++ APIs provide a **login** method. The method in the blocking C API is called **SMB_login** and the method 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 return value of these methods provides the handle to the instance. For more information, see Information About API Initialization.

- Blocking API Example
- Non-Blocking API Example

Blocking API Example

The following C++ blocking API **logoutByName** method signature:

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

is translated into C blocking API as:

```
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 into C non-blocking API as:

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

Information About API Initialization

To initialize the API

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

The following sections describe these steps in more detail.

- API Construction
- Setup Operations
- Connecting to the Subscriber Manager

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 has succeeded before proceeding.

To construct and initialize the C++ API, construct and API object and call the **init** function as shown in the following example:

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

To construct and initialize the C API, call the **init** function, which allocates and initializes the API.

Example:

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

Setting the LEG Name

Set the Login Event Generator (LEG) name if you intend to turn on the SM-LEG failure handling options in the SM. For more information about LEGs and SM-LEG failure handling, see the *Cisco SCMS Subscriber Manager User Guide* .

To set the LEG name, call the relevant **setName** function in the API. The SM will use the LEG name when recovering from a connection failure. A constant string that identifies the API will be appended to the LEG name as follows:

- Blocking API: **.B.SM-API.C**
- Non-blocking API: **.NB.SM-API.C**

Setting the LEG Name 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 SM-LEG failure handling, see *Appendix A* of the *Cisco SCMS Subscriber Manager User Guide*.

Setup Operations

The setup operations for the two APIs differ. Both APIs support setting a disconnect listener, which is described in Information About the Disconnect Callback Listener.

The following sections describe the setup operations for the blocking API and the non-blocking API.

- Blocking API Setup
- Non-blocking API Setup

Blocking API Setup

To set up the blocking API, you must set an operation timeout value. For more information, see the Blocking API module.

Non-blocking API Setup

To set up the non-blocking API you are required to set a disconnect callback, see the Non-blocking API module.

Connecting to the Subscriber Manager

The following example shows how to connect to the SM 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.

Information About the 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 API allocates the return code structure and the API user must free the structure. You can use the **freeReturnCode** utility function to safely free the structure.

Additional return code structure utility functions are:

- **printReturnCode** —Prints the **ReturnCode** structure value to the stdout
- **isReturnCodeError** —Checks whether the **ReturnCode** structure is an error

Definitions

From the **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;
```

Return Code Structure 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. This example code uses the **isReturnCodeError** and **freeReturnCode** methods.

```
ReturnCode* subFields = bapi.getSubscriber("subscriber1");
if (isReturnCodeError(subFields) == false)
{
```

```
printf("\tname:\t\t%s\n", subFields->u.objectArray[0]->u.stringVal);
printf("\tmapping:\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);
}
else
{
printf("error in subscriber retrieval\n");
}
freeReturnCode(subFields);
```

Information About the 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

Error Code Structure Definition

From the **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.

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

Information About Network ID Mappings

A network ID mapping is a network identifier that the SCE device can relate to a specific subscriber record. A typical example of a network ID mapping (or simply mapping) is an IP address. For additional information, see the *Cisco SCMS Subscriber Manager User Guide* . Currently, the Cisco Service Control Solution supports IP address, IP range, and VLAN mappings.

Both blocking and non-blocking APIs contain operations that accept mappings as a parameter. Examples consist of the following:

- **addSubscriber** operation (blocking APIs)

- **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 number 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 **GeneralDefs.h** header file provides the mapping type of an IP address:

- **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 is a decimal in the specified range.

The **GeneralDefs.h** header file also provides the mapping type.

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

Subscriber Domains

A domain is an identifier that tells the SM which SCE devices to update with the subscriber record. For more information about domains, see the *Cisco SCMS Subscriber Manager User Guide*.

A domain name is of type (**char***). During system installation, the network administrator determines the system domain names, which therefore vary between installations. The APIs include methods that specify to which domain a subscriber belongs and which also allow queries about the system 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.

The automatic domain roaming feature of the SM allows subscribers tomove between domains by calling the login method for a subscriber with an updated domain parameter.

Note Automatic domain roaming is not compatible with previous versions of the SM API, because these versions did not allow changing the domain of the subscriber.

Subscriber Properties

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

More information about properties can be found in the *Cisco SCMS Subscriber Manager User Guide* and in the *Cisco Service Control Application for Broadband (SCA BB) User Guide*. The latter provides application-specific information. It lists the subscriber properties that exist in the application running on your system, the allowed value set, and the significance of each property value.

To format subscriber properties for C/C++ API operations, use the String arrays (**char****) **propertyKeys** and **propertyValues**.

Note The arrays must be of the *same length* , and NULL entries are forbidden. Each key in the keys array has a matching entry in the values array. The value for **propertyKeys[j]** resides in **propertyValues[j]**.

Example:

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

Custom Properties

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

To format custom properties, use the string (**char****) arrays **customPropertyKeys** and **customPropertyValues,** the same as used in formatting Subscriber Properties.

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 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.h** header file provides the **Logger** class.

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 **setLogger** method. The **PrintLogger.h** header file provides the **PrintLogger** class.

Information About the Disconnect Callback Listener

Both blocking and non-blocking APIs allow setting a disconnect callback listener to be notified when the API is disconnected from the SM. The disconnect callback is defined as follows:

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

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

Disconnect Callback Listener Example

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

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

Signal Handling

The SCMS SM C/C++ 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.

To ignore the signal, add the following call:

```
sigignore(SIGPIPE);
```

Practical Tips

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

- Connect to the SM once and maintain an open API connection to the SM at all times, using the API many times. Establishing a connection is a timely procedure, which allocates resources on the SM side and the API client side.
- Share the API connection between your threads. It is better to have one connection per LEG. Multiple connections require more resources on the SM and client side.
- Do not implement synchronization of the calls to the API. The client automatically synchronizes calls to the API.
- It is recommended to place the API clients (LEGs) in the same order of the SM machine processor number.
- If the LEG application has bursts of logon operations, enlarge the internal buffer size accordingly to hold these bursts (Non-blocking flavor).
- During the integration, set the SM **logon_logging_enabled** configuration parameter to view the API operations in the SM log to troubleshoot the integration, if any problems arise.
- Use the debug mode for the LEG application that logs/prints the return values of the non-blocking operations.
- Use the automatic reconnect feature to improve the resiliency of the connection to the SM.
- In cluster setups, connect the API using the virtual IP address of the cluster and not the management IP address of one of the machines.

CHAPTER 3

Blocking API

This module describes the features and operations of the blocking API and provides code examples. It also introduces the Reply Timeout, a feature unique to the blocking API.

Note If you only need to develop an *automatic integration*, skip this module and go directly to Chapter 4, "Non-blocking API."

- Information About Multi-threading Support
- Operation Timeout Error Code
- Information About Blocking API Methods
- Blocking API C++ Code Examples
- Blocking API C Code Examples

Information About Multi-threading Support

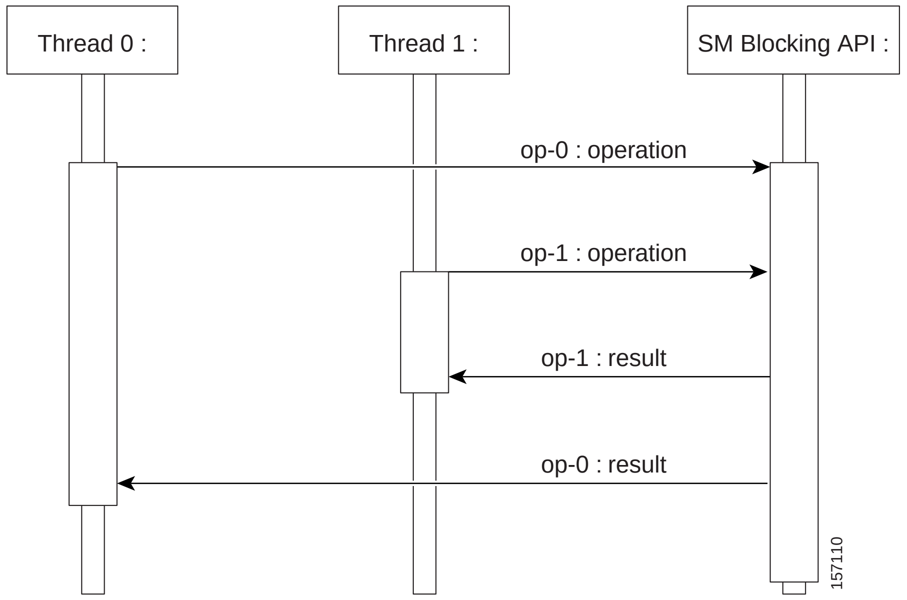
The blocking API supports a configurable number of threads calling its methods simultaneously. For more information about configuring the number of threads, see the "C++ init Method" section on page 3-31.

Note In a multi-threaded scenario for the blocking API, the order of invocation is **not** guaranteed.

Multi-threading Support 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):

Figure 3-1 Multi-threading Support



The SM allocates five threads to handle each API instance. Cisco Systems recommends that you develop a multi-threaded application that uses the API with a number of threads in the order of the five threads. Implementing with more threads might result in longer delays for the calling threads.

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. However, the SM API provides means of working around this situation, using the reply timeout feature.

The reply timeout feature, or **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.

Information About Blocking API Methods

This section lists the methods of the blocking API. A description of input parameters and return values for each method follows its syntax.

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 functions and syntax structure.

The C/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. The function description explains any other differences between the APIs.

The blocking API subscriber management methods can be classified into the following categories:

- **Dynamic IP and property allocation** —For using the SM API for integration with an AAA system, the following methods are relevant:
  - 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.

- **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, disconnect:**
  - C++ setLogger Method

  - C++ init Method
  - C SMB_init Function
  - C SMB_release Function
  - setReconnectTimeout
  - setName
  - connect
  - disconnect
  - isConnected

Note The examples that appear at the end of the described methods are in C++. Every example described at the end of the methods should be preceded by the following sample code:

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

- login
- logoutByName
- logoutByNameFromDomain
- logoutByMapping
- loginCable
- logoutCable
- addSubscriber
- removeSubscriber
- removeAllSubscribers
- getNumberOfSubscribers
- getNumberOfSubscribersInDomain
- getSubscriber
- subscriberExists
- subscriberLoggedIn
- getSubscriberNameByMapping
- getSubscriberNames
- getSubscriberNamesInDomain
- getSubscriberNamesWithPrefix
- getSubscriberNamesWithSuffix
- getDomains
- setPropertiesToDefault
- removeCustomProperties
- C++ setLogger Method

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 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.

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

argMappingsSize —The size of the **argMappings** and **argMappingTypes** arrays.

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

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

argPropertySize —The size of the **argPropertyKeys** and **argPropertyValues** arrays.

argDomain —See explanation of Subscriber Domains.

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

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

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 has occurred.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION
- ERROR_CODE_UNKNOWN

  The following can cause this error:

  – NULL value for the **domain** parameter for the subscriber that does not exist/does not have a domain

  – Invalid values for the **propertyValues** parameter

For a description of error codes, see List of Error Codes.

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
- Description
- Parameters
- Return Value
- Error Codes
- Example

Syntax

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

Description

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

Parameters

argName —See explanation of Subscriber Name Format.

argMappings —See explanation of mappings and mapping types in the Information About Network ID Mappings section. If no mappings are specified, all the subscriber mappings will be removed.

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

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

Error Codes

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

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

Example

To remove IP address 192.168.12.5 from 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 from subscriber *john* :

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

logoutByNameFromDomain

- Syntax
- Description
- Parameters
- Return Value
- Error Codes
- Example

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 the subscriber.

Parameters

argName —See explanation of Subscriber Name Format.

argMappings —See explanation of mappings and mapping types in the Information About Network ID Mappings section. If no mappings are specified, all the subscriber mappings will be removed.

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

argMappingsSize —The size of the **argMappings** and **argMappingTypes** arrays.

argDomain —See explanation of Subscriber Domains.

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

Error Codes

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

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

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
- Description
- Parameters
- Return Value
- Error Codes
- Example

Syntax

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

Description

Locates a subscriber based on domain and mapping, and removes the subscriber mappings. The subscriber remains in the database.

Parameters

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

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

argDomain —See description in the Parameters section of the logoutByNameFromDomain method..

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

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

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

Example

To remove IP address 192.168.12.5 from domain **subscribers** :

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

loginCable

- Syntax
- Description
- Parameters
- Return Value
- Error Codes
- Examples

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, which calls the cable support module in the SM. This method logs 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 *Cisco 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 Subscriber Domains.

The domain will usually be CMTS IP.

Note Domain aliases must be set on the SM in order to correctly interpret the CMTS IP as a domain name. For information regarding aliases configuration, see the "Configuring Domains" section of *Cisco SCMS Subscriber Manager User Guide* .

argPropertyKeys —See explanation of the keys and values in the Subscriber Properties section. If the CPE is provided with partial or no application properties, the values for the missing application properties will be copied from the application properties of the CM to which this CPE belongs. Each CM application property thus serves as a default for the CPE under it.

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

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
- Description
- Parameters
- Return Value
- Error Codes
- Examples

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 Parameters section of the loginCable method.

argCm —See description in the Parameters section of the loginCable method.

argIp —See description in the Parameters section of the loginCable method.

argDomain —See description in the Parameters section of the loginCable 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* that is behind *CM1* :

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

addSubscriber

- Syntax
- Description
- Parameters
- Return Value
- Error Codes
- Example

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 the first time the subscriber is created by using **addSubscriber**.

Note With **addSubscriber** , the auto-logout feature is always disabled. To enable auto-logout, use **login**.

Example:

Subscriber *AB* , already set up in the subscriber database, has a single IP mapping of *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 a **login** operation with these NULL-value parameters will not change *AB* 's mappings; *AB* will still have its previous IP mapping of *IP1* .

Parameters

argName —See explanation of Subscriber Name Format.

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

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

argMappingsSize —The size of the **argMappings** and **argMappingTypes** arrays.

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

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

argPropertySize —The size of the **argPropertyKeys** and **argPropertyValues** arrays.

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

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

argPropertySize —The size of the **argCustomPropertyKeys** and **argCustomPropertyValues** arrays.

argDomain —See explanation of Subscriber Domains.

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

Return Value

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

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION

- ERROR_CODE_UNKNOWN

  This error code indicates that invalid values were supplied for the **propertyValues** parameter.

For a description of error codes, see List of Error Codes.

Example

To add a new subscriber, *john* , with 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
- Description
- Parameters
- Return Value
- Error Codes
- Example

Syntax

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

Description

Removes a subscriber completely from the SM database.

Parameters

argName —See explanation of Subscriber Name Format.

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; i.e., the subscriber was not found in the database, or the subscriber was found but was not successfully removed.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

Example

To remove subscriber *john* entirely from the database:

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

removeAllSubscribers

- Syntax
- Description
- Return Value
- Error Codes

Syntax

```
ReturnCode* removeAllSubscribers()
```

Description

Removes all subscribers from the SM, leaving the database with no subscribers.

Note This method might 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
- Description

- Return Value
- Error Codes

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
- Description
- Parameters
- Return Value
- Error Codes

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

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

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE _DOMAIN_NOT_FOUND

For a description of error codes, see List of Error Codes.

getSubscriber

- Syntax
- Description
- Parameters
- Return Value
- Error Codes
- Example

Syntax

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

Description

Retrieves a subscriber record. Each field is formatted as an integer, string, or string array, as described 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.

Return Value

A pointer to a **ReturnCode** structure holding an array of **ReturnCode** structures with nine elements. No array element is NULL.

The following list is the element values and their meanings:

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

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

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

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
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

Verifies that a subscriber exists in the SM database.

Parameters

argName —See explanation of Subscriber Name Format.

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
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

Checks whether a subscriber that already exists in the SM database is logged in; i.e., if the subscriber also exists in an SCE database.

When the SM is configured to work in *Pull mode* , a TRUE value returned by this method does **not** guarantee that the subscriber actually exists in an SCE database, but rather 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.

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

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

getSubscriberNameByMapping

- Syntax
- Description
- Parameters
- Return Value
- Error Codes

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 in the Information About Network ID Mappings section.

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

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

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

Return Value

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

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

- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

getSubscriberNames

- Syntax
- Description
- Parameters
- Return Value
- Error Codes
- Example

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

Note 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 lower value of **argAmount** and the SM limit (1000) limits the array size.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

Example

To receive an alphabetical list of subscriber names:

```
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
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

Retrieves subscribers from the SM database that are associated with the specified domain.

The function of this operation is the same as the getSubscriberNames operation.

Parameters

argFirstName —See description in the Parameters section of the getSusbcriberNames operation.

argAmount —See description in the Parameters section of the getSusbcriberNames 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 specified domain.

See the Return Value section of the getSusbcriberNames operation for more information.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _DOMAIN_NOT_FOUND
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

getSubscriberNamesWithPrefix

- Syntax
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

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

The function of this operation is the same as the getSubscriberNames operation.

Parameters

argFirstName —See description in the Parameters section of the getSusbcriberNames operation.

argAmount —See description in the Parameters section of the getSusbcriberNames 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 the Return Value section of the getSusbcriberNames operation for more information.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

getSubscriberNamesWithSuffix

- Syntax
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

Retrieves subscribers from the SM database whose names end with the specified suffix.

The function of this operation is the same as the getSubscriberNames operation.

Parameters

argFirstName —See description in the Parameters section of the getSusbcriberNames operation.

argAmount —See description in the Parameters section of the getSusbcriberNames 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 the Return Value section of the getSusbcriberNames operation for more information.

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

getDomains

- Syntax
- Description
- Return Value
- Error Codes

Syntax

```
ReturnCode* getDomains()
```

Description

Provides a 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.

setPropertiesToDefault

- Syntax
- Description
- Parameters
- Return Value
- Error Codes

Syntax

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

Description

Resets the specified application properties of a subscriber. If an application is installed, the relevant application properties will be set to the default value of the properties according to the currently loaded application information. If an application is not installed, an **ERROR_CODE_ILLEGAL_STATE** error code is returned.

Parameters

argName —See explanation of Subscriber Name Format.

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

argPropertySize —The size of the **argPropertyKeys** array.

Return Value

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

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

removeCustomProperties

- Syntax
- Description
- Parameters
- Return Value
- Error Codes

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.

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

argCustomPropertySize —The size of the **argCustomPropertyKeys** array.

Return Value

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

Error Codes

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

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see List of Error Codes.

C++ setLogger Method

- Syntax
- Description
- Parameters
- Return Value

Syntax

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

Description

Sets an implementation of the abstract Logger class. Use this method to integrate the SM API log messages with the host application log.

Parameters

argLogger —An implementation of the abstract Logger class.

Return Value

None.

C++ init Method

- Syntax
- Description
- Parameters
- Return Value
- Example

Syntax

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

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).

argReconnectTimeout —When the connection to the SM is down, the API will attempt to re-establish the connection after this timeout.

Return Value

Boolean value:

- TRUE—Success
- FALSE—Fail

Example

```
SmBlockingApi bapi;
bool success = bapi.init(10,
0,
2000000, //default
10,      //default
20,      //default
0);      //default (no reconnect)
```

C SMB_init Function

- Syntax
- Description
- Parameters
- Return Value
- Example

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 —The number of threads the API should support.

argThreadPriority —The priority for the PRPC protocol network thread.

argBufferSize —The internal buffer size (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

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
- Description
- Parameters
- Return Value

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.

setReconnectTimeout

- Syntax
- Description
- Parameters
- Return Value

Syntax

```
void setReconnectTimeout(Uint32 reconnectTimeout)
```

Description

Sets the reconnection timeout.

When the connection to the SM is down, the API will attempt to re-establish the connection after 'reconnection timeout' seconds.

Parameters

reconnectTimeout —The timeout.

Return Value

None.

setName

- Syntax
- Description
- Parameters
- Return Value

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
- Description
- Parameters
- Return Value

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
- Description
- Return Value

Syntax

```
bool disconnect()
```

Description

Attempts to terminate the PRPC protocol connection to the SM.

Return Value

Boolean value:

- TRUE—Success
- FALSE—Fail

isConnected

- Syntax
- Description
- Return Value

Syntax

```
bool isConnected();
```

Description

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

Return Value

Boolean value:

- TRUE—The connection is up.
- FALSE—The connection is down.

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
- Adding a Subscriber, Printing Information, Removing a 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* numberOfSusbcribersInDomain=
bapi.getNumberOfSubscribersInDomain(
domains->u.stringArrayVal[i]);
if (isReturnCodeError(numberOfSusbcribersInDomain) == false)
{
printf("number of susbcribers domain %s:\t\t%d\n",
domains->u.stringArrayVal[i],
numberOfSusbcribersInDomain->u.intVal);
}
freeReturnCode (numberOfSusbcribersInDomain);
}
}
freeReturnCode (domains);
freeReturnCode (totalSubscribers);
//finalization
bapi.disconnect();
return 0;
}
```

Adding a Subscriber, Printing Information, Removing a Subscriber

The following program adds a subscriber to the subscriber database, retrieves the subscriber information, prints it to stdout, and 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 = "10";
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%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 a Subscriber, Printing Information, Removing a 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* numberOfSusbcribersInDomain=
SMB_getNumberOfSubscribersInDomain(bapi,
domains->u.stringArrayVal[i]);
if(isReturnCodeError(numberOfSusbcribersInDomain) == false
{
printf("number of susbcribers domain %s:\t\t%d\n",
domains->u.stringArrayVal[i],
numberOfSusbcribersInDomain->u.intVal);
}
freeReturnCode (numberOfSusbcribersInDomain);
}
}
freeReturnCode (domains);
freeReturnCode (totalSubscribers);
//finalization
SMB_disconnect(bapi);
SMB_release(bapi);
return 0;
}
```

Adding a Subscriber, Printing Information, Removing a Subscriber

The following program adds a subscriber to the subscriber database, retrieves the subscriber information, prints it to **stdout** , and 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 = "10";
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%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");
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);
}
}
```

CHAPTER 4

Non-blocking API

This module describes the features and operations of the Non-blocking API and provides code examples.

This module introduces the Result Handler Callbacks, a feature unique to the Non-blocking API.

- Multi-threading Support
- Information About Result Handler Callbacks
- Information About Non-blocking API Methods
- Non-blocking API C++ Code Examples
- Non-blocking API C Code Examples

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.

Information About Result Handler Callbacks

The Non-blocking API enables the setting of 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,
ReturnCode *argReturnCode);
/* operation success callback specification */
typedef void (*OperationSuccessCallBackFunc)(Uint32 argHandle,
ReturnCode *argReturnCode);
```

You should implement these callbacks if you want to be informed about the success or 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**.
- Result Handler Callbacks Example

Result Handler Callbacks Example

The following example is a simple implementation of a result handler that counts the number of success/failure operations. This main method initiates 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;
}
```

Information About 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 Information About Result Handler Callbacks). If an internal error 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, except that: return values of **Void** are translated to **NULL**.

Note The error/fail callback will be handed with an error **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. The function description explains any other differences between the APIs.

The following methods are described:

- login
- logoutByName
- logoutByNameFromDomain
- logoutByMapping
- loginCable
- logoutCable
- C++ setLogger Method
- C++ init Method
- C SMNB_init Function
- C SMNB_release Function
- setReconnectTimeout
- setName
- connect
- disconnect
- isConnected

login

- Syntax

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

logoutByName

- Syntax

Syntax

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

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

logoutByNameFromDomain

- Syntax

Syntax

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

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

logoutByMapping

- Syntax

Syntax

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

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

loginCable

- Syntax

Syntax

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

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

logoutCable

- Syntax

Syntax

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

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

C++ setLogger Method

- Syntax

Syntax

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

The operation functionality is the same as the matching Blocking API operation. See the C++ setLogger Method operation for more information.

C++ init Method

- Syntax

Syntax

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

The operation functionality is the same as the matching Blocking API operation. See the C++ init Method operation for more information.

C SMNB_init Function

- Syntax
- Return Value

Syntax

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

The operation functionality is the same as the matching Blocking API operation. See the C SMB_init Function operation for more information.

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

Syntax

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

The operation functionality is the same as the matching Blocking API operation. See the C SMB_release Function operation for more information.

setReconnectTimeout

- Syntax

Syntax

```
void setReconnectTimeout(Uint32 reconnectTimeout)
```

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

setName

- Syntax

Syntax

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

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

connect

- Syntax

Syntax

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

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

disconnect

- Syntax

Syntax

```
bool disconnect()
```

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

isConnected

- Syntax

Syntax

```
bool isConnected()
```

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

Non-blocking API C++ Code Examples

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

- Login and Logout

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-susbcribers>");
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 provides a code example for logging in and logging out subscribers.

- Login and Logout

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-susbcribers>");
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);
}
```

APPENDIX A

List of Error Codes

This module provides a list of error codes that are used in the C/C++ API.

- List of Error Codes

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. The following table gives a list of the error codes and their descriptions.

Table A-1 List of Error Codes

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

Table A-1 List of Error Codes

Error Code	Description
`ERROR_CODE_SUSBSCRIBER_ALREADY_EXISTS`	The subscriber on which the operation was performed already exists in the SM database.
`ERROR_CODE_DATABASE_EXCEPTION`	Internal SM error – database error occurred during the operation.
`ERROR_CODE_ARRAY_ACCESS`	Internal SM error.
`ERROR_CODE_ATTRIBUTE_NOT_FOUND`	Internal SM error.
`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.